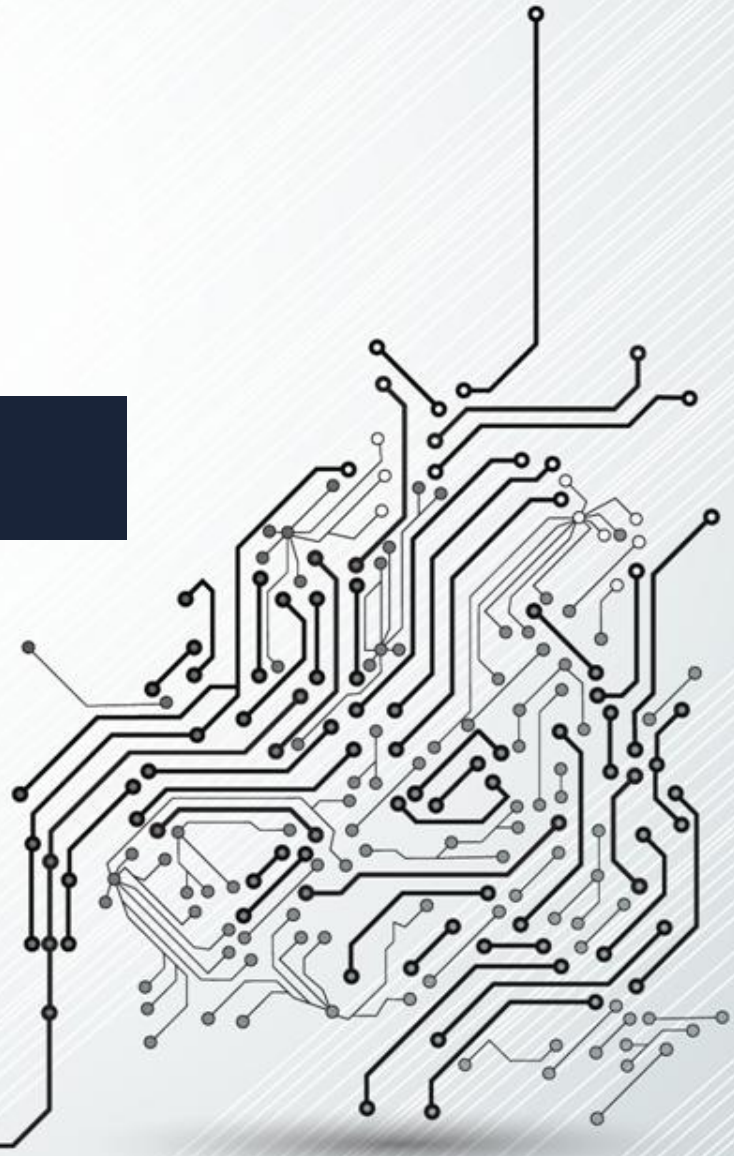
Smartphone Hacking

The Art of Android Exploitation

By: Malachi Jones, PhD



DARK LABS

Booz | Allen | Hamilton

About Me

https://www.linkedin.com/in/malachijonesphd

- **Education**
  - Bachelors Degree: Computer Engineering (Univ. of Florida, 2007)
  - Master's Degree: Computer Engineering (Georgia Tech, 2009)
  - PhD: Computer Engineering (Georgia Tech, 2013)
- **Cyber Security Experience**
  - Harris: Cyber Software Engineer (2013-2014)
  - Harris: Vulnerability Researcher (2015)
  - Booz Allen DarkLabs : Embedded Security Researcher (2016- Present)

About Dark Labs

Booz Allen Dark Labs is an elite team of security researchers, penetration testers, reverse engineers, network analysts, and data scientists, dedicated to stopping cyber attacks before they occur.[1]

(1 http://darklabs.bah.com)

2015: A Year of Embedded Exploitation

Jeep PWNED

(Link)

2015: A Year of Embedded Exploitation

(Link)

2015: A Year of Embedded Exploitation

(Link)

2016: The Exploitation Continues…

PWNED

[(Link)]

Outline

- Motivation: Understanding both sides (offense & defense)
- Objectives of Talk
- Review
  - Secure Coding
  - Vulnerability Research (VR)
- Vulnerability Discovery on Android
- Weaponizing & Deploying Exploits
- Demo/Conclusion

Motivation: Understanding Offense & Defense

- Defensive Cyber Awareness
  - Understanding and implementation of current best practices (e.g. secure coding) to build a hardened security system
  - Successfully making tradeoffs between security, performance, and cost to develop a system that is useful, secure, and affordable

Motivation: Understanding Offense & Defense

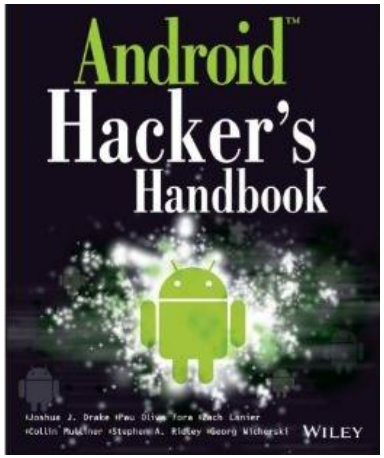
- Offensive Cyber Awareness
  - Forefront of new attacks & techniques
  - Can help defenders develop mitigation techniques before a new attack can cause significant damage
  - ***Ability to think like the adversary to find and patch potential vulnerabilities before they do***

Objectives of Talk

- Discuss techniques (borrowed from hackers) that can allow the defender to find and patch vulnerabilities before a hacker does

- Demonstrate how effortless it can be to weaponize a vulnerability to pown devices.

Android Hacker's Handbook



- ***This presentation is based on material from the Handbook***
- **Joshua Drake** is the lead author and also the person who discovered the Android vulnerability dubbed *StageFright*
- Handbook provides a lot more in-depth coverage on Android Hacking than we can cover in this presentation

Secure Coding

Review

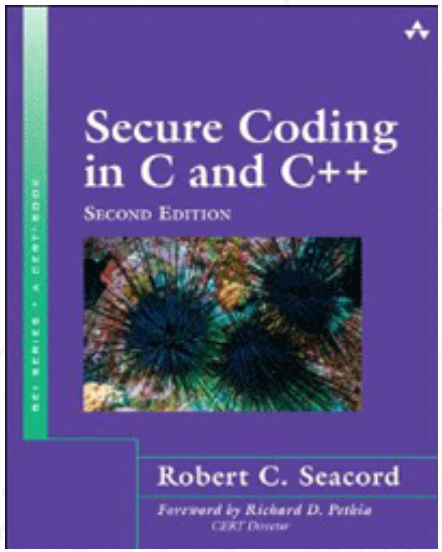
Review: Secure Coding

- Low level code (e.g. drivers and kernels) of embedded devices such as *smartphones* are written in C/C++
- The genius/curse of C/C++
  - *Trust the programmer*
  - *Don't prevent the programming from doing what needs to be done*
  - Make it fast, even if it is not guaranteed to be portable
- ***How does this translate to security and safety?***

It doesn't…..

Review: Secure Coding

- Best practices for secure C/C++ coding written by Robert C. Seacord (*CERT @ CMU)*
- Based on common mistakes that inexperienced and professional software developers make
- Should be a mandatory reading for any aspiring C/C++ programmer



Review: Secure Coding

- **Question***:* So, how can bugs still manifest even if best practices such as secure coding principles are adhered to?
- **Answer*:*** An attacker can exploit bugs in the interactions (external and/or internal) of components in hardware and/or software systems

- **Case Study**: *Hot Potato Exploit* (link)
  - Affects all Windows versions including Windows 7 & 8
  - Exploits bugs in the interaction with legacy software components that include NetBIOS Name Service (NBNS)
  - Can enable remote code execution by an attacker

Vulnerability Research

Review

Review: Vulnerability Research

- Primary objective of **Vulnerability Research (VR)** is to find exploitable bugs (not all bugs are exploitable)
- Often, a **proof of concept (PoC)** is desirable that demonstrates the extent to which the vulnerability can be weaponized in a *reliable* way to pown devices.

Review: Vulnerability Research

- Complications (in the exploitation of bug)
  - **Address Space Layout Randomization (ASLR)** : Randomizes base address of modules; *difficult for attacker to know where to direct cpu to execute code*
  - **Data Execution Prevention (DEP)** : Makes non-code memory regions not executable; *no shell code execution*
  - **Reliability**: The vulnerability may be difficult to consistently reproduce; *vulnerability may require the system to be in an unlikely state for exploitation*

Review: Vulnerability Research

- VR Process Overview
  - **Profiling Target**: Information gathering which includes port scanning and preliminary reviews of device services and security features (e.g. ASLR and SELinux)
  - **Initial Access:** Preferably obtaining access to a shell or a debug serial port on a test device that is similar to target device
  - **Vulnerability Discovery:** Activities include static analysis, dynamic analysis, and fuzzing to discover an exploitable bug
  - **Exploitation/Weaponization**: Development of a proof-of-concept that demonstrates the exploit capabilities of bug.

Review: Vulnerability Research

- VR Process Overview
  - **Profiling Target**: Information gathering which includes port scanning and preliminary reviews of device services and security features (e.g. ASLR and SELinux)
  - **Initial Acces**[illegible] to a shell or a debug serial port on a test device that is similar to target device
  - **Vulnerability Discovery:** Activities include static analysis, dynamic analysis, and fuzzing to discover an exploitable bug
  - **Exploitation/Weaponization**: Development of a proof-of-concept that demonstrates the exploit capabilities of bug.

Presentation Focus

Review: Vulnerability Research

- Vulnerability Discovery
  - **Static Analysis:** Analyzing code and data in the application without directly executing the application; may require converting the binary from machine code to C-like code, aka decompiling.
  - **Dynamic Analysis:** Executing the application in an instrumented or monitored manner to garner more concrete information on behavior
  - **Fuzzing:** Dynamic method for testing software input validation by feeding it intentionally malformed input

Review: Vulnerability Research

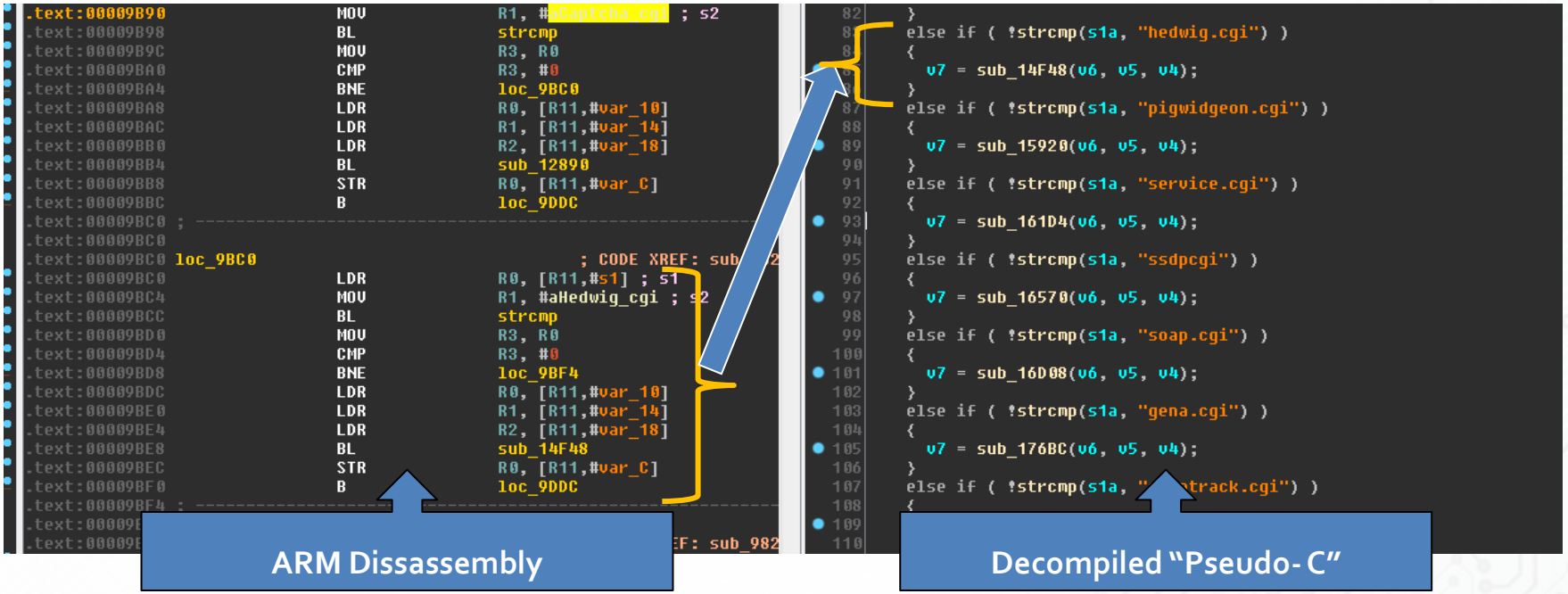
```
bool validatePassword(char * usercredential, int password_len, int username_len)
{
char username[100];
char password[100];

//Get username
memcpy(username, usercredential,username_len);

char *p_password =  usercredential+ username_len +1;

//Get password
memcpy(password, p_password,password_len);

……
}
```

Password buffer has length of 100

No check to make sure that the password length does not exceed the local buffer size of '100'

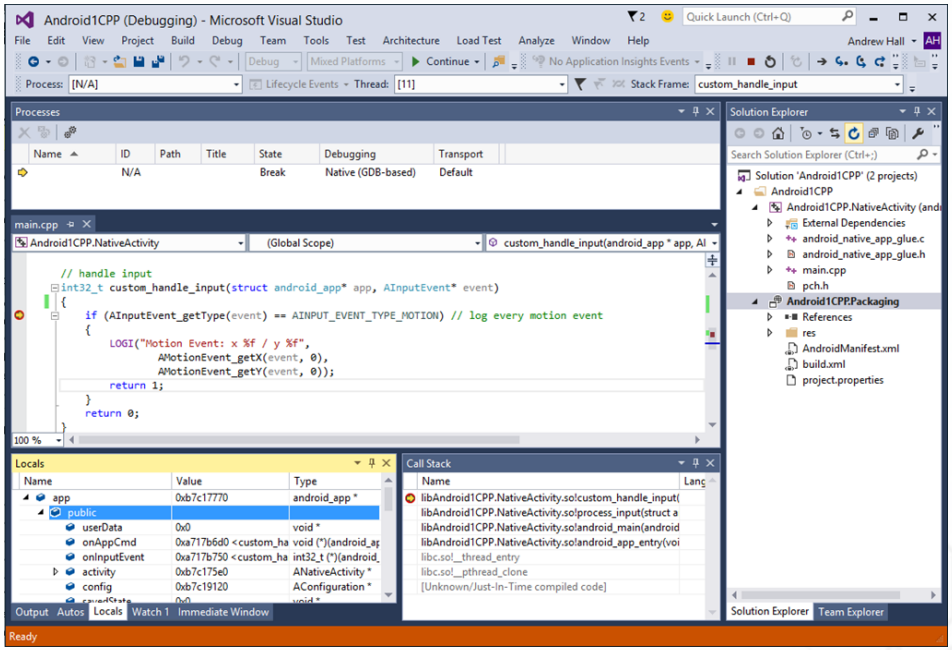
- **Static Analysis**
  - In the above example, there is a stack overflow vulnerability
  - By passing in a 'password' with a length greater than 100, possible to overflow buffer, which can allow for control of code execution

Review: Vulnerability Research



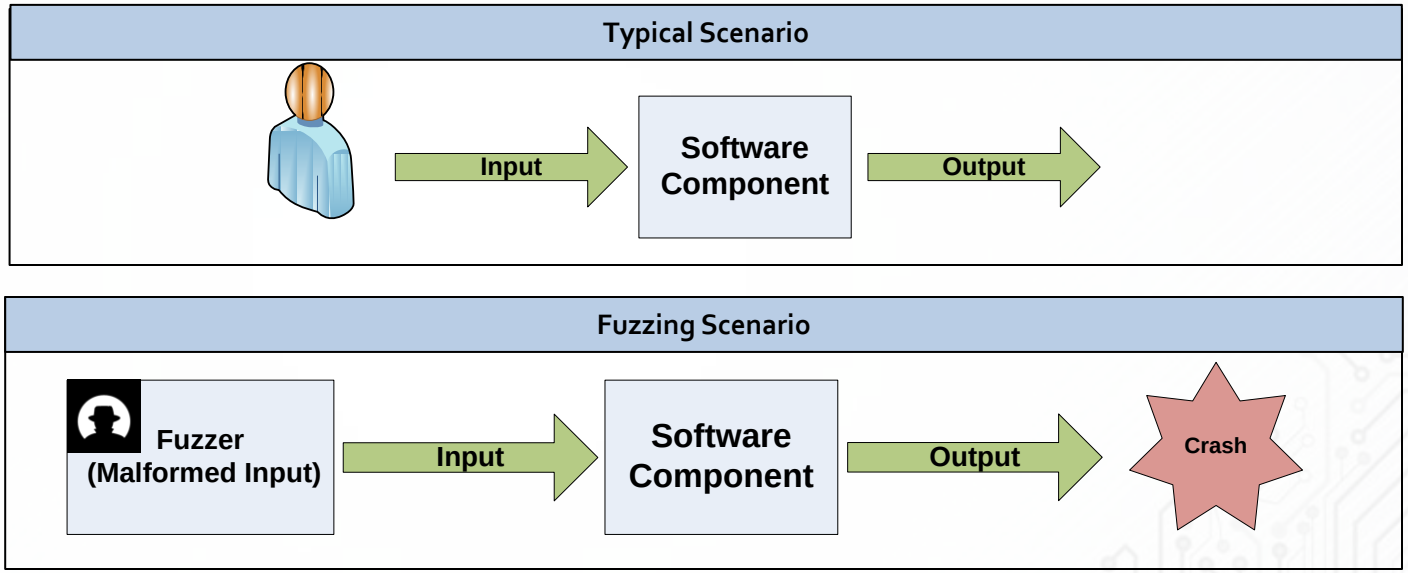
- **Static Analysis**
  - If the source is unavailable, then it may be necessary to decompile the binary into psuedo code
  - Tools such as IDA PRO can be used to aid in reversing

Review: Vulnerability Research



- **Dynamic Analysis**
  - Can be used to step through the code to understand behavior
  - Variables and software state can be manipulated

Review: Vulnerability Research

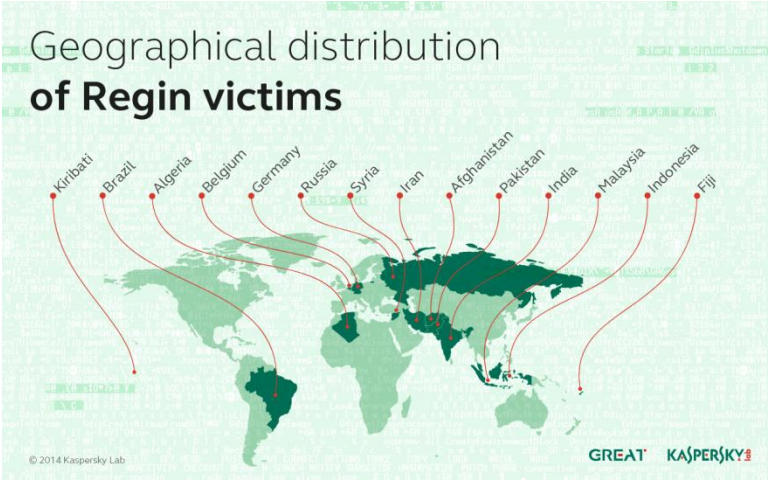


- **Fuzzing**
  - A primary objective is to generate malformed input that will induce unexpected behavior in software that can include a crash
  - Additional static/dynamic analysis can then be used to determine root cause and whether or not the bug is exploitable

Review: Vulnerability Research

- Exploitation/Weaponization
  - **Remote Access:** Setting up a backdoor on the victim that allows for future command and control capabilities
  - **Phoning home state of victim:** Periodically updating the host about the current activities and state of victim
  - **Persistence:** Includes surviving a reboot of victim and attempts to remove installed '*custom*' software

Review: Vulnerability Research



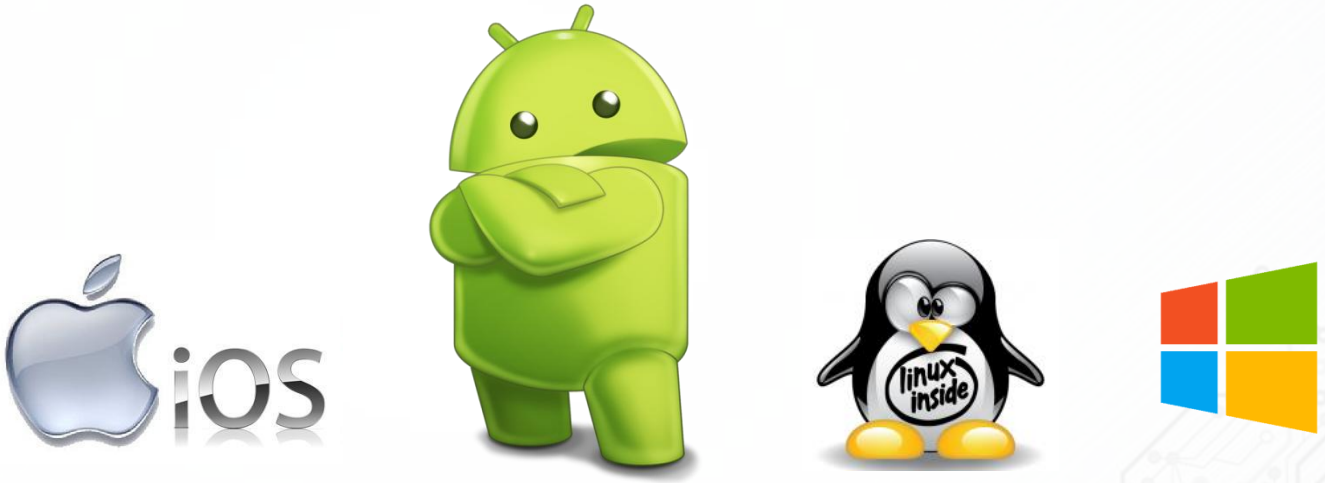
- **Weaponization**: *Case Study of Regin (link)*
  - **Remote Access:** Cyberattack platform that is deployed on victim networks for total remote control of host at all levels
  - **Phoning Home State of Victim**: Forms a peer-to-peer network with other infected machines to send information back to the command and control center
  - **Persistence**: Covertly installs kernel modules and drivers

Review: Vulnerability Research

- **Question**: Are Hackers the only ones that conduct vulnerability research? If not, why would others do this type of research?
- **Answer**:
  - ***The motivation behind this presentation is that the defenders should also be using VR techniques to identify exploitable bugs.***
  - Some companies have bug bounty hunting programs, where a white/grey hat identifies a bug in exchange for money
  - Others hire VR personnel onsite to hack into their own stuff. Example: Charlie Miller and Chris Valasek were hired by *Uber* after their *Chrysler* car hack demo (link).

Vulnerability Discovery on Android

Vulnerability Discovery on Android



- **Important Note**: *Although concepts and techniques discussed in this section are targeted at the Android platform, they are also relevant for other platforms including iOS, Linux, and Windows.*

Vulnerability Discovery on Android

Overall Goal:

Exercise code on the target with malformed inputs that we generate

Vulnerability Discovery on Android

- Steps for Exploit Discovery using Fuzz Testing
  1. Choose a software target
  2. Generate Inputs
  3. Delivering & Processing Inputs
  4. Monitor for crashes

Vulnerability Discovery on Android (The Steps)

1) Choose a Software Target

Vulnerability Discovery: Choosing a Target

- **Objective 1.1**: *Find a software attack surface whose code can be exercised **remotely** by an adversary*




Vulnerability Discovery: Choosing a Target

- We'll target the Android browser

- **Why?**
  - Standard on all Android devices
  - Since web browsers focus on performance, much of the code is implemented in C/C++
  - Very complex with support for new technologies that include HTML5, and the browser is essentially a mini OS

Vulnerability Discovery: Choosing a Target

- **Objective 1.2**: *Select a web technology that the browser implements to focus on*

- **Note:** One of the most challenging aspects of fuzzing is to determining where to focus the fuzzing efforts

Vulnerability Discovery: Choosing a Target

- We'll focus on the `Typed Array` feature of *HTML5*

- **Why *HTML5***
  - Relatively new technologies, which means there are bound to be undiscovered bugs
  - Has a rich feature set that includes support for video and audio, which means high complexity;complexity often introduces bugs

Vulnerability Discovery: Choosing a Target

- **Background**: `Typed Arrays`
  - Allows web developers access to a region of memory that is formatted as a native array
  - *Example Snippet*:

```
var arr = new Uint8Array(16);
for (var n = 0; n < arr.length; n++)
{
   arr[n] = n
}
```

  - *Snippet Explained* : Creates an array of 16 elements and initializes them from 0 to 15

Vulnerability Discovery: Choosing a Target

```
var   arr = new Uint8Array(16);
for (var n = 0; n < arr.length; n++)
{
   arr[n] = n
}
```

- **Key Concepts:**
  - The typed array is a ***native array***, which means that it doesn't need to be translated between JavaScript and native representation
  - Translating back and forth between JavaScript and native representation impacts performance
  - ***Browser can achieve greater performance through improved efficiency using this type of array***

Vulnerability Discovery: Choosing a Target

```
var   arr = new Uint8Array(16);
for (var n = 0; n < arr.length; n++)
{
   arr[n] = n
}
```

- Previously disclosed bugs in Typed Array :
  - Researcher Pinkie Pie compromises of Chrome @ 2013 Mobile pwn2Own (link)
  - Researcher geohot compromises Chrome @ 2014 pwn2Own (link)

Vulnerability Discovery on Android (The Steps)

2) Generating Inputs

Vulnerability Discovery: Generate Inputs

- So now that we have our target, we'd like to generate inputs to see how the target handles it
- Below is an example snippet that would generate different 'types' of Typed Arrays

```
def generate_var():
    vtype = random.choice(TYPEDARRAY_TYPES)
    vlen  = rand_num()
    return "var array1 = new %s(%d);" % (vtype, vlen)
```

  - **Note**: *Generating 'good' inputs is not trivial*. It requires an understanding of the target and the possible unaddressed corner cases and boundary conditions

Vulnerability Discovery on Android (The Steps)

3) Delivering & Processing Inputs

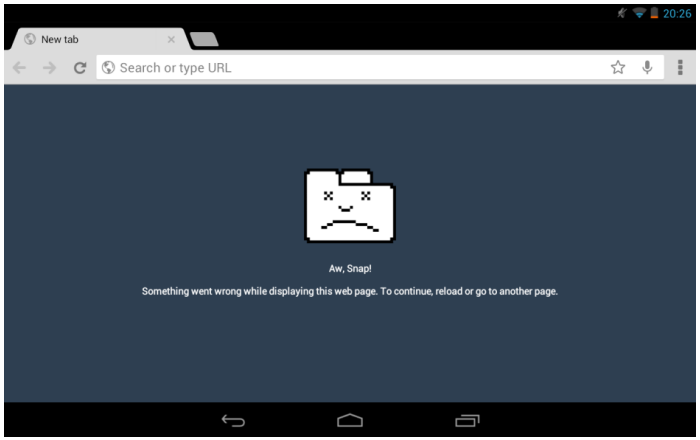
Vulnerability Discovery: Delivering Inputs

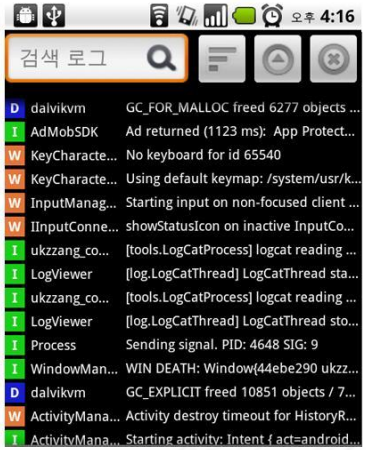
- **Delivering Inputs**
  - What we'd like is a way to get the inputs to the target and have the target execute it
  - Tools such as `BrowserFuzz` have lightweight http servers that can serve up the input to the target
- **Processing Input**
  - Getting chrome to process our special URL (that points to our fuzz html server) can be automated
  - We can take advantage of the `ActivityManager` to start the browser and tell it where to load content from via `Intents`

Vulnerability Discovery on Android (The Steps)

4) Monitoring for Crashes

Vulnerability Discovery: Monitoring for Crashes





- **Key Concept:** *Monitoring the behavior of the target program is essential to knowing whether you've discovered something that is noteworthy*
- When a process on android crashes, `debuggerd` writes information about the crash to the system log
- So we can monitor when the browser crashes by checking the system log.

Weaponizing and Deploying Exploits

Weaponizing and Deploying Exploits

- So we've found an exploit… How do we make it useful?

Weaponizing and Deploying Exploits

- The Steps (*Browser exploit*)
  1. **Choose** the target
  2. **Develop** or **utilize** an exploit delivery mechanism (e.g. Metasploit)
  3. **Get in-between** the victim and the network
  4. **Inject** exploit code in the victim's HTML traffic
  5. **Implant** malware (e.g. key logger, screenscraper, etc..)

Weaponizing and Deploying Exploits(The Steps)

1) Choose the target

Weaponizing and Deploying Exploits

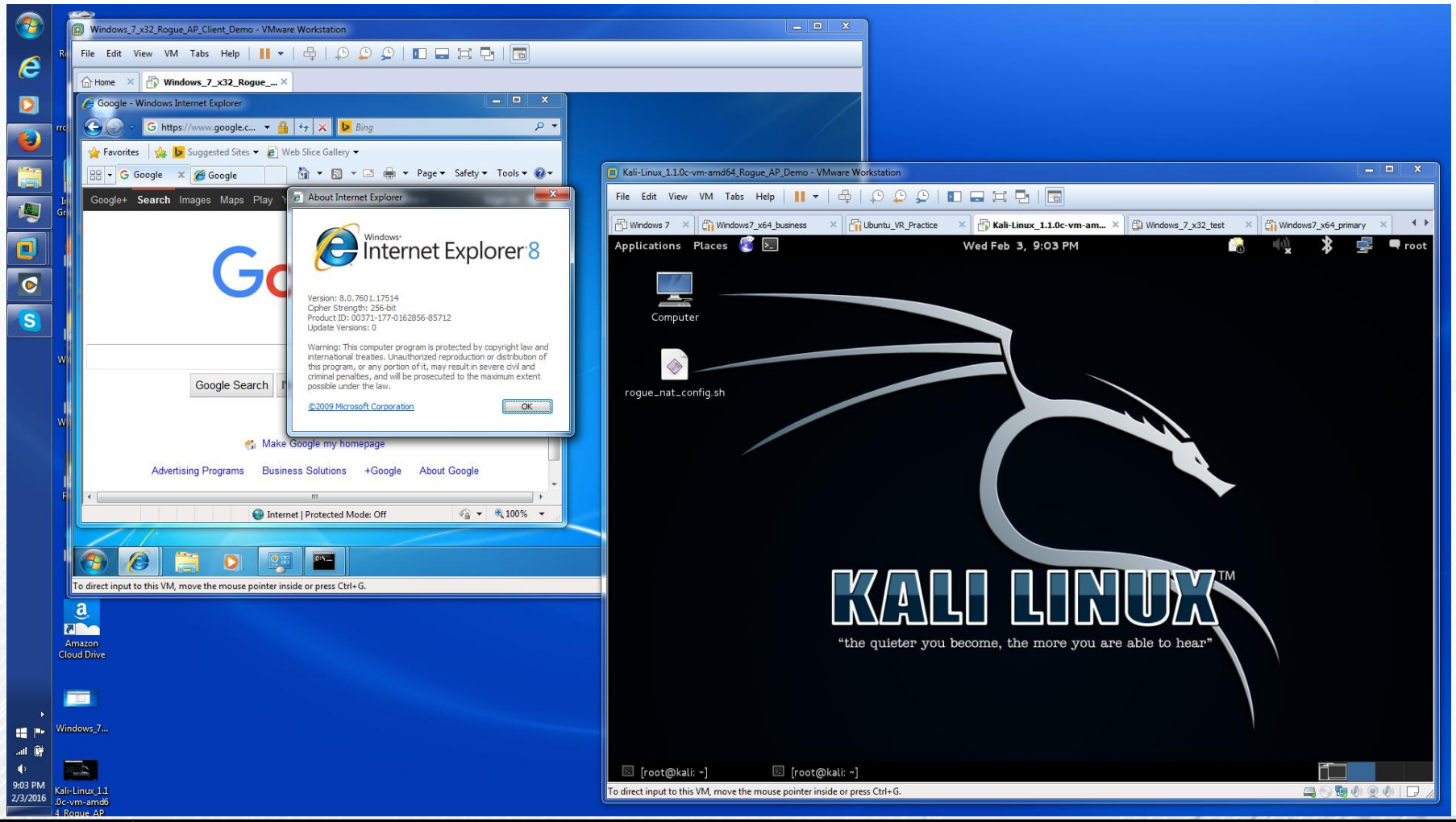
- **Possible Targets for Browser Exploit**

- **Note:** *This exploit technique can be applied to any computing device (e.g. in-car entertainment system) that uses a browser*

Weaponizing and Deploying Exploits

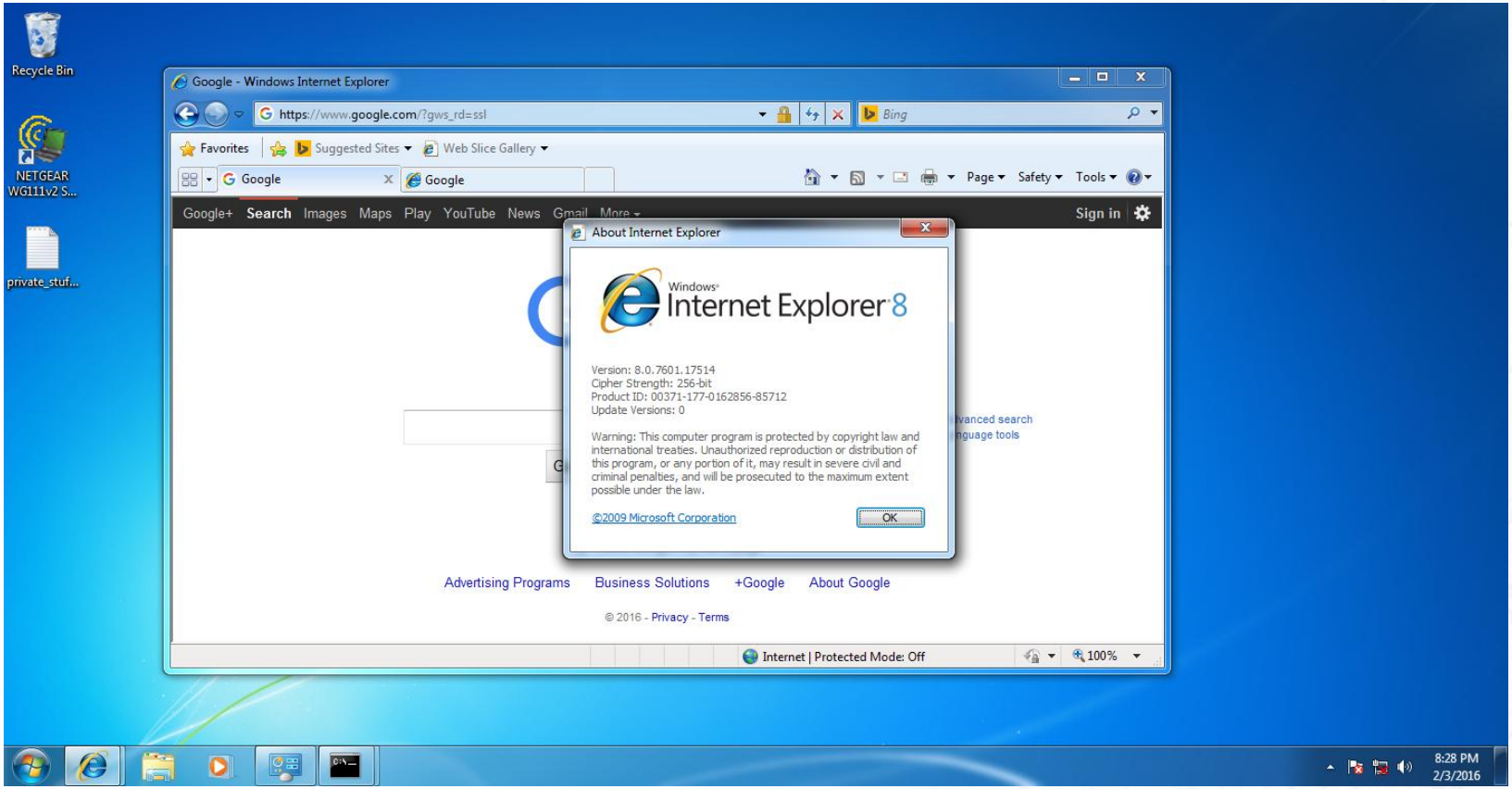
- **For convenience, we'll target a laptop**

Weaponizing & Deploying: The VM Setup



Choose the target

- We'll target IE 8 (because there is a nice metasploit package for it)

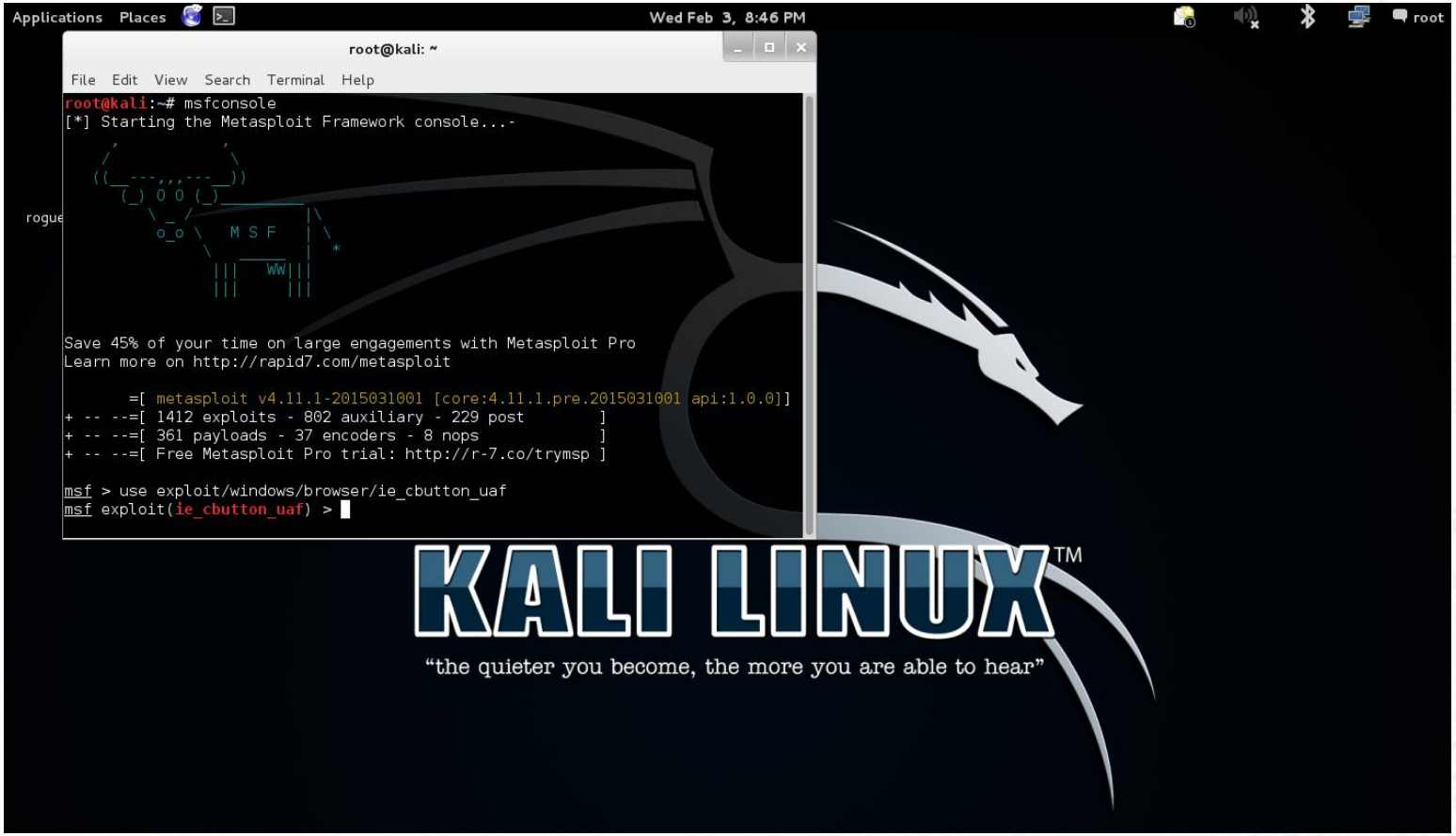


Weaponizing and Deploying Exploits(The Steps)

2) Develop or utilize an exploit delivery mechanism

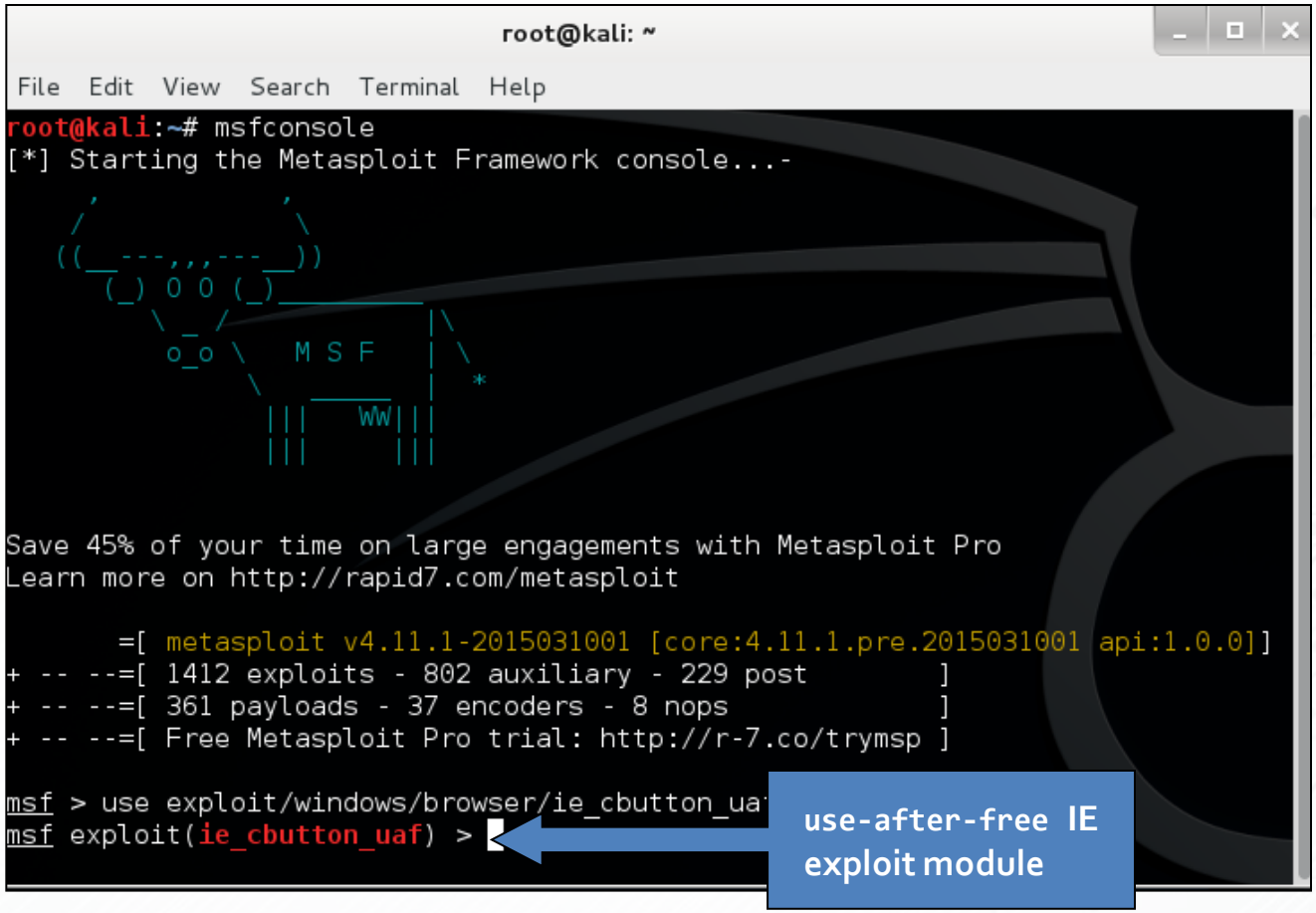
Exploit delivery mechanism

- Metasploit will be our exploit delivery mechanism



Exploit delivery mechanism

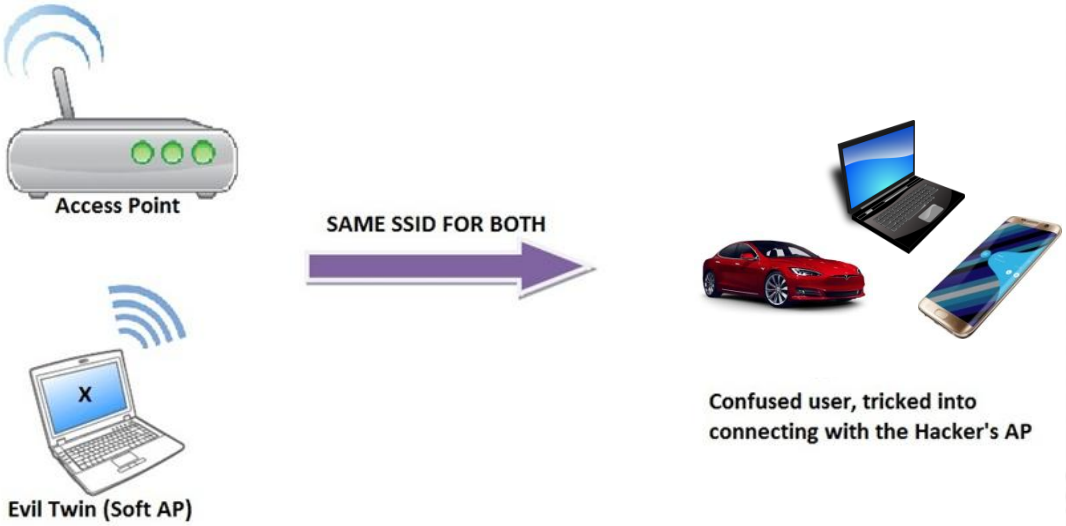
- Note the IE exploit `ie_cbutton_uaf` (link)



Weaponizing and Deploying Exploits(The Steps)

3) Get in-between the victim and the network

Man-in-the-middle target device



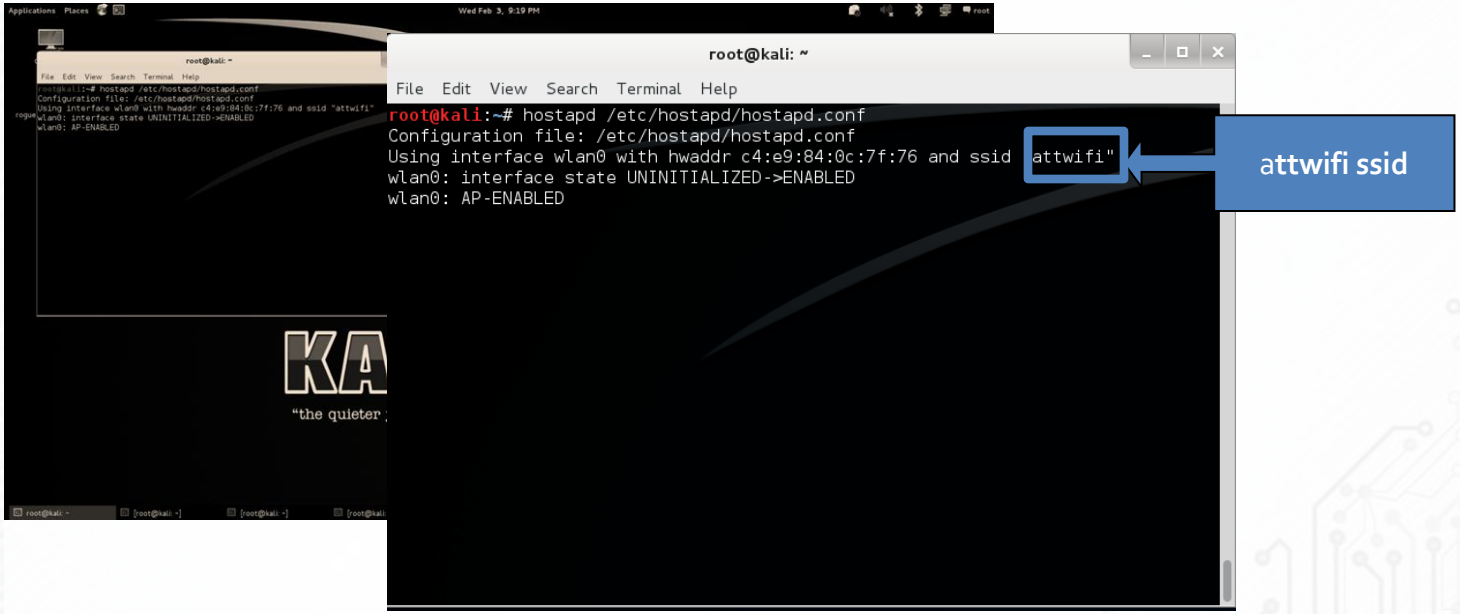
- **Evil Twin Attack** (link)
  - We'll setup a rogue access point with the `ssid` *attwifi*
  - Most smartphones have attwifi as an `ssid` that they will automatically try to connect to it

Man-in-the-middle target device

TP-LINK TL-WN722N

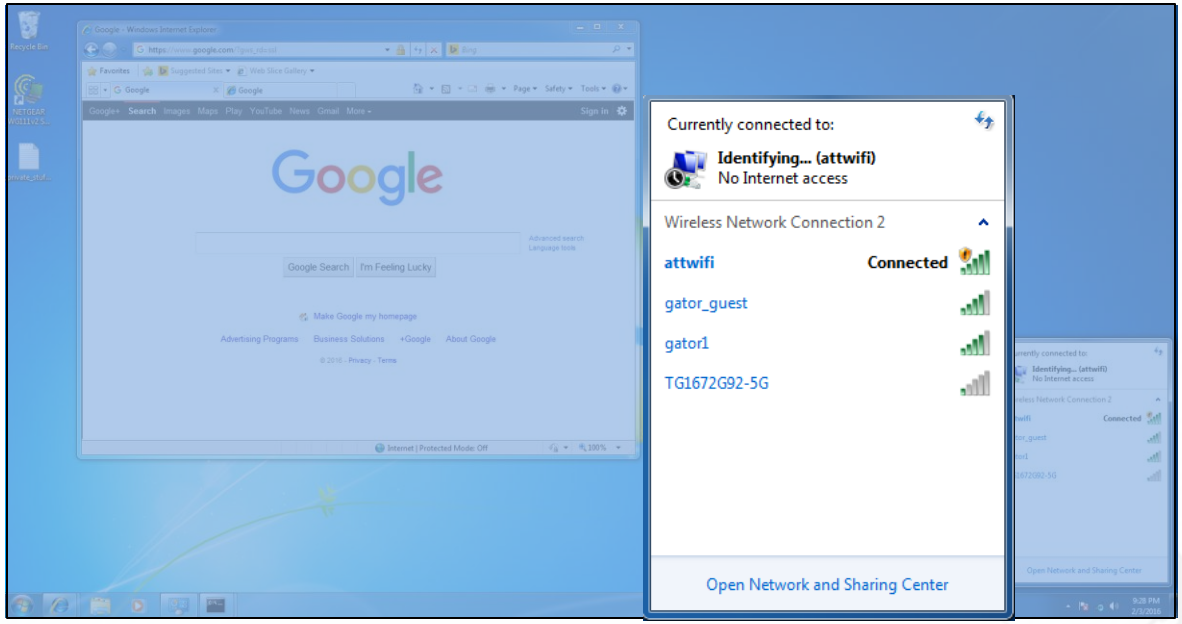
- **Hardware for Evil Twin Attack**
  - We'll need a wifi adapter that supports master mode
  - **`master`** mode enables adapter to function as an AP
  - The TP-Link WN722N supports **`master`** mode and other modes that include **`monitor`** mode

Man-in-the-middle target device



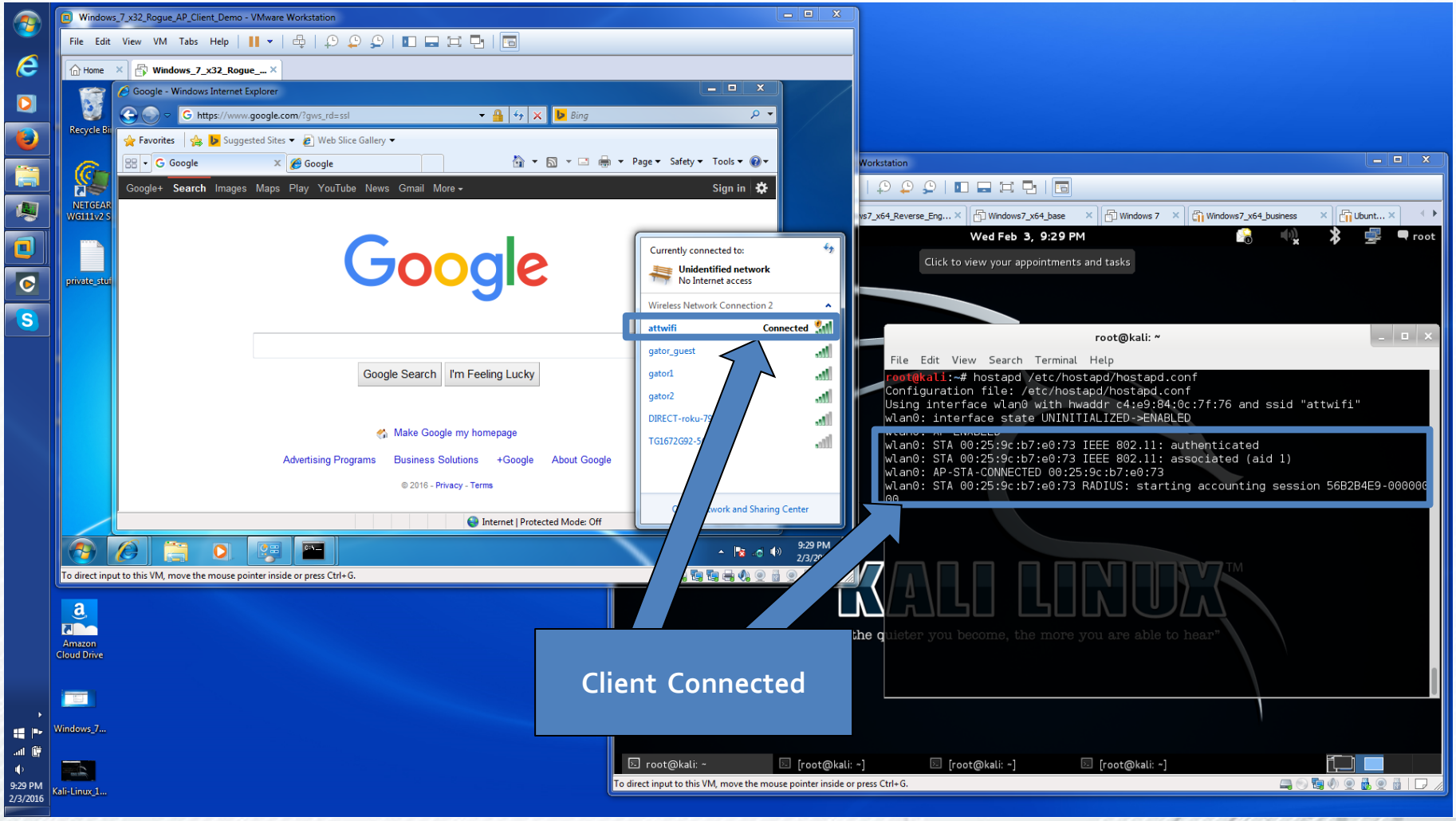
- **Software for Evil Twin Attack**
  - `hostapd` allows for a software AP to be created
  - Tool supports features including WPA

Man-in-the-middle target device



- **Software for Evil Twin Attack**
  - hostapd allows for a software AP to be created
  - Tool supports features including WPA

Man-in-the-middle target device

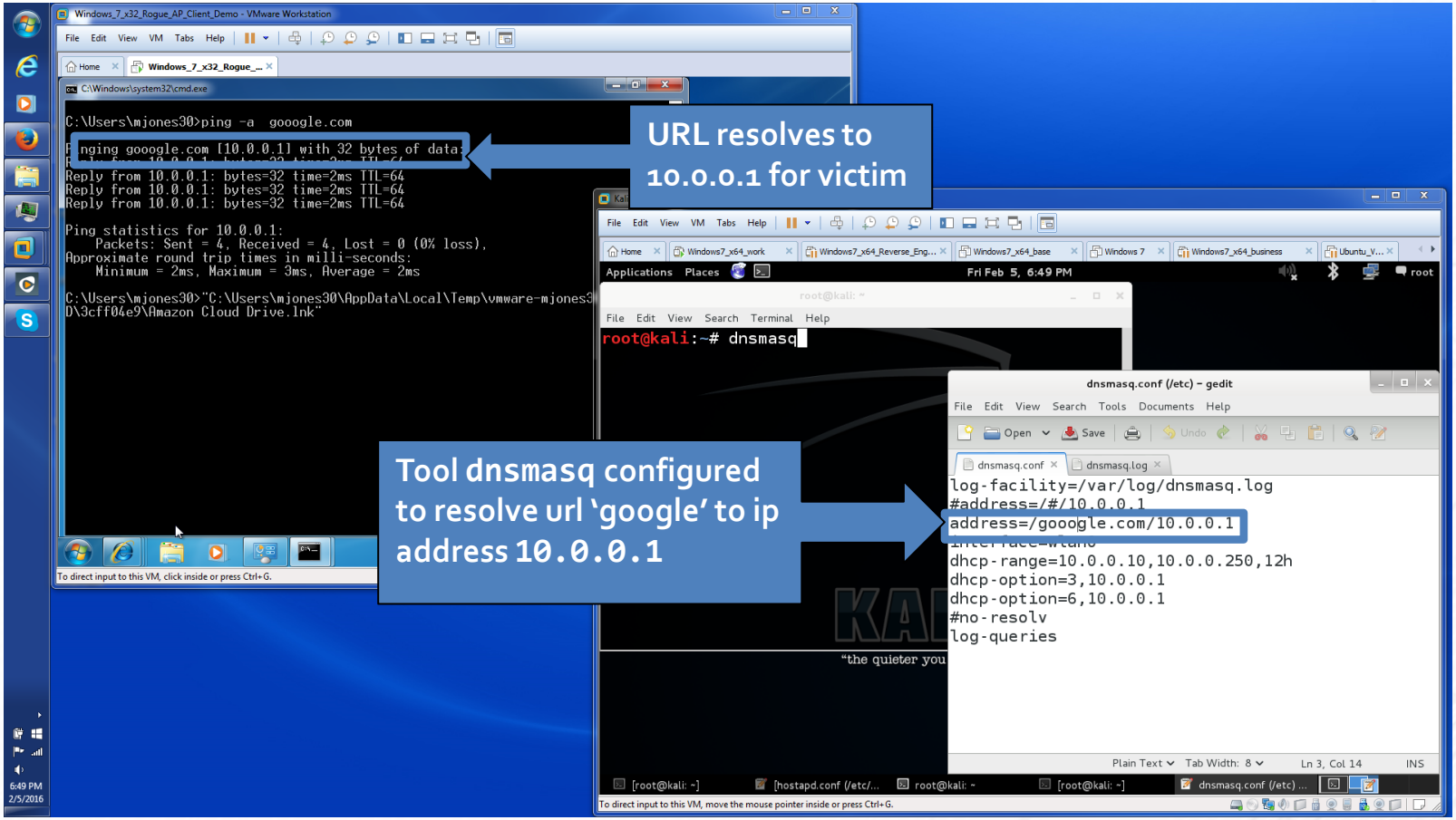


Weaponizing and Deploying Exploits(The Steps)

4) Inject exploit code in the victim's HTML traffic

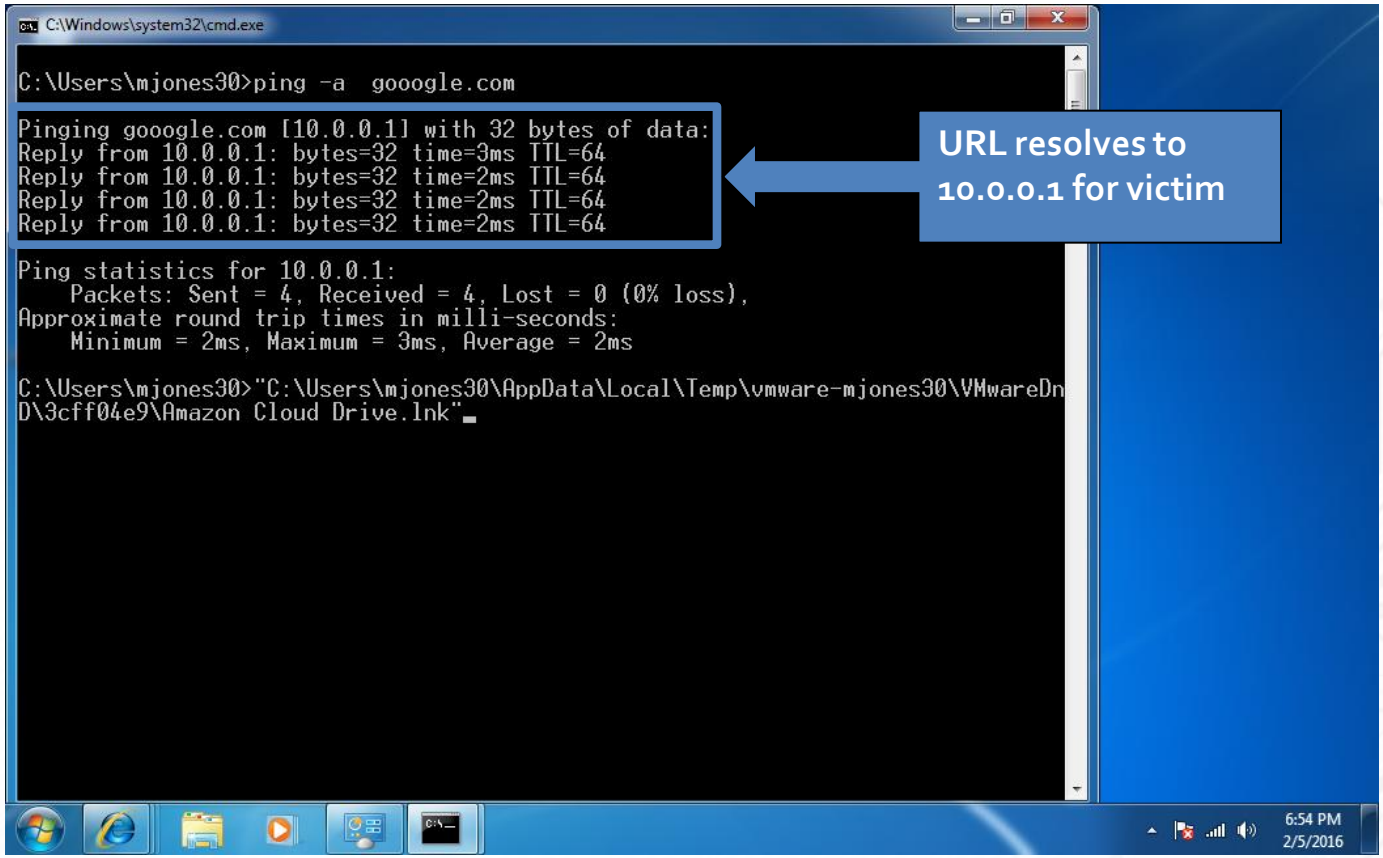
Inject Exploit Code

- 4a : DNS injection → url "gooogle.com" → exploit server 10.0.0.1"



Inject Exploit Code

- 4a : DNS injection → url "gooogle.com" → exploit server 10.0.0.1"

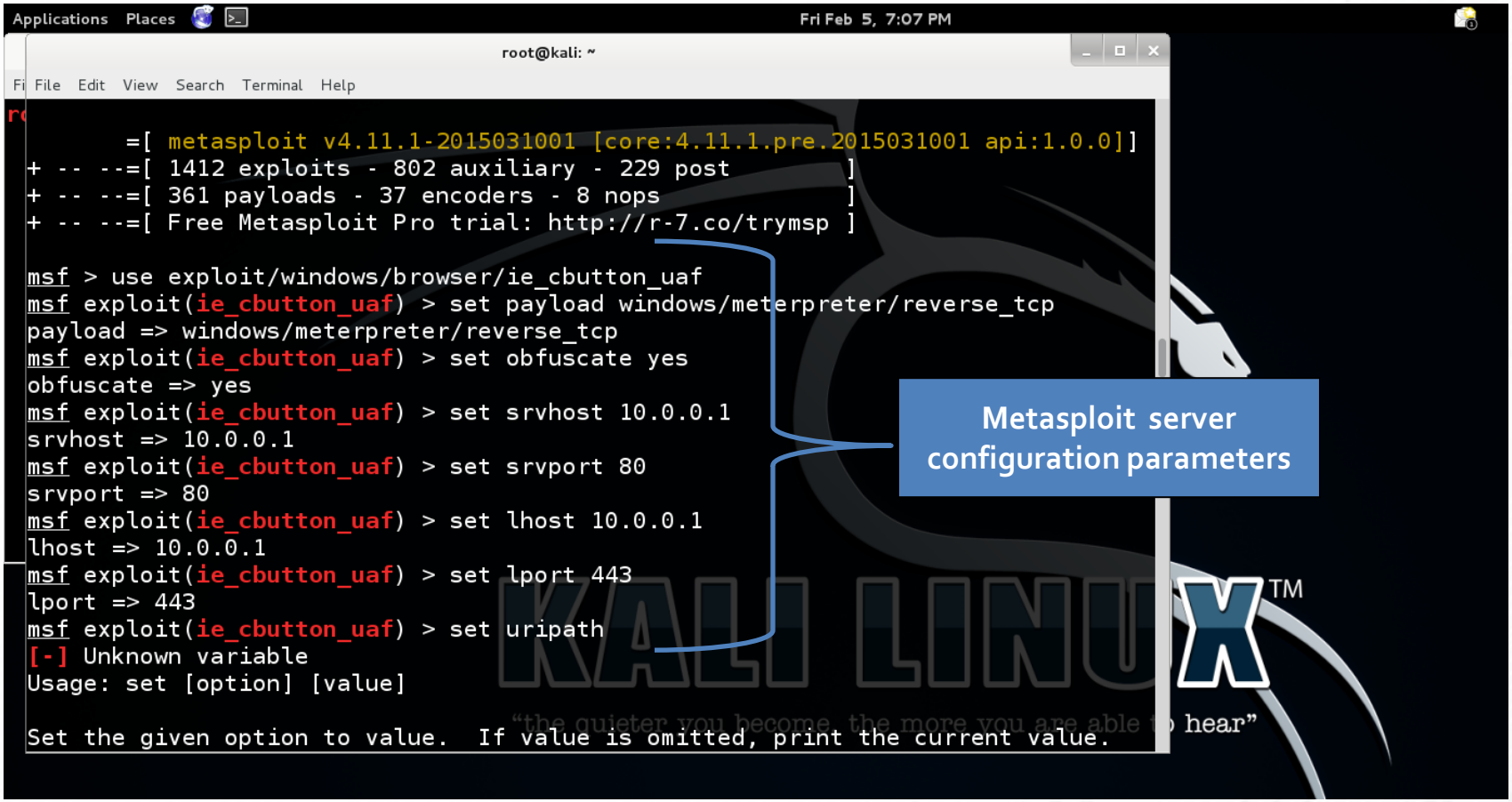


Inject Exploit Code

- 4b : Serve up a modifed `gooogle.com` webpage to victim

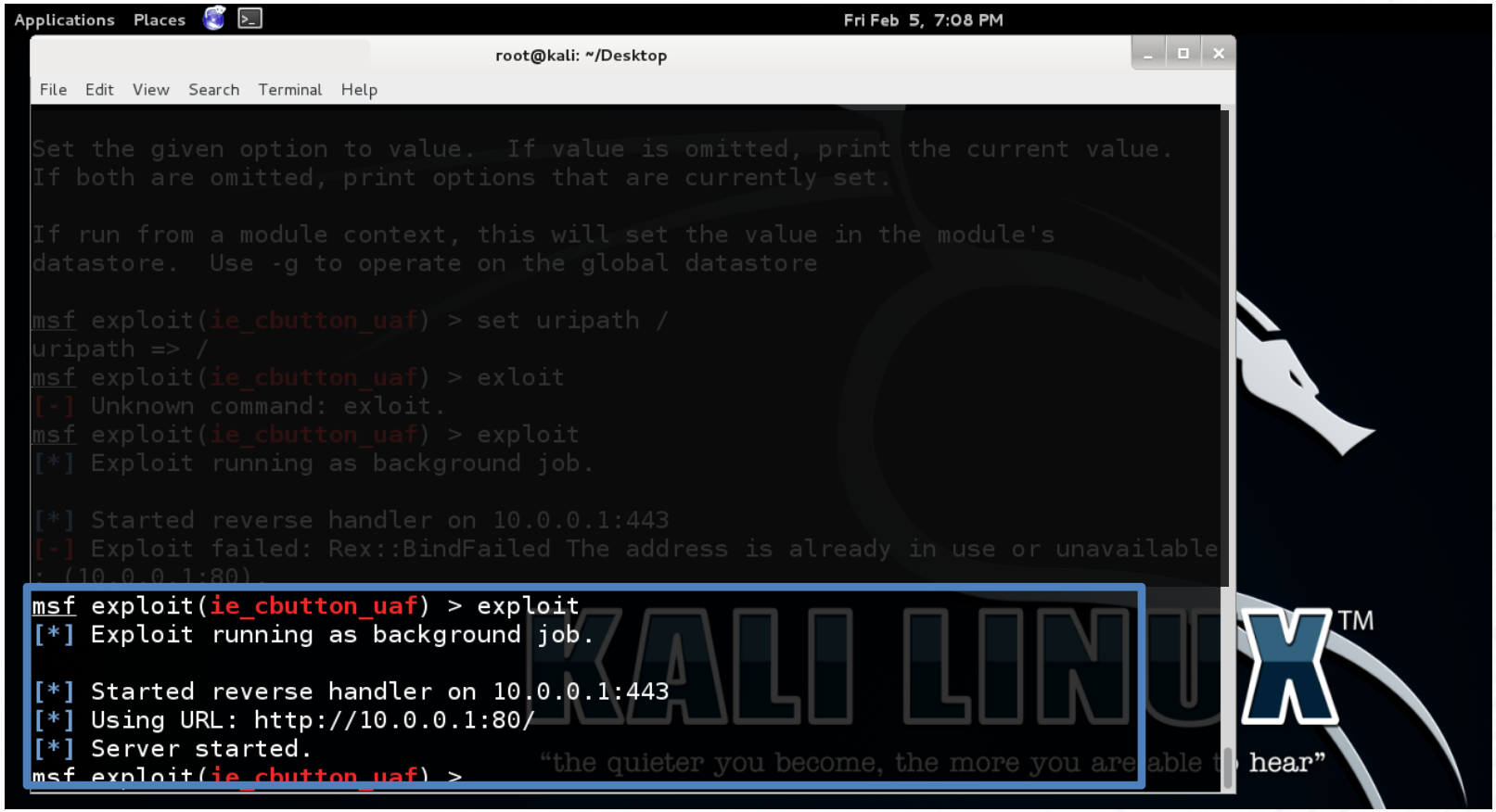
Inject Exploit Code

- 4c : Configure metasploit



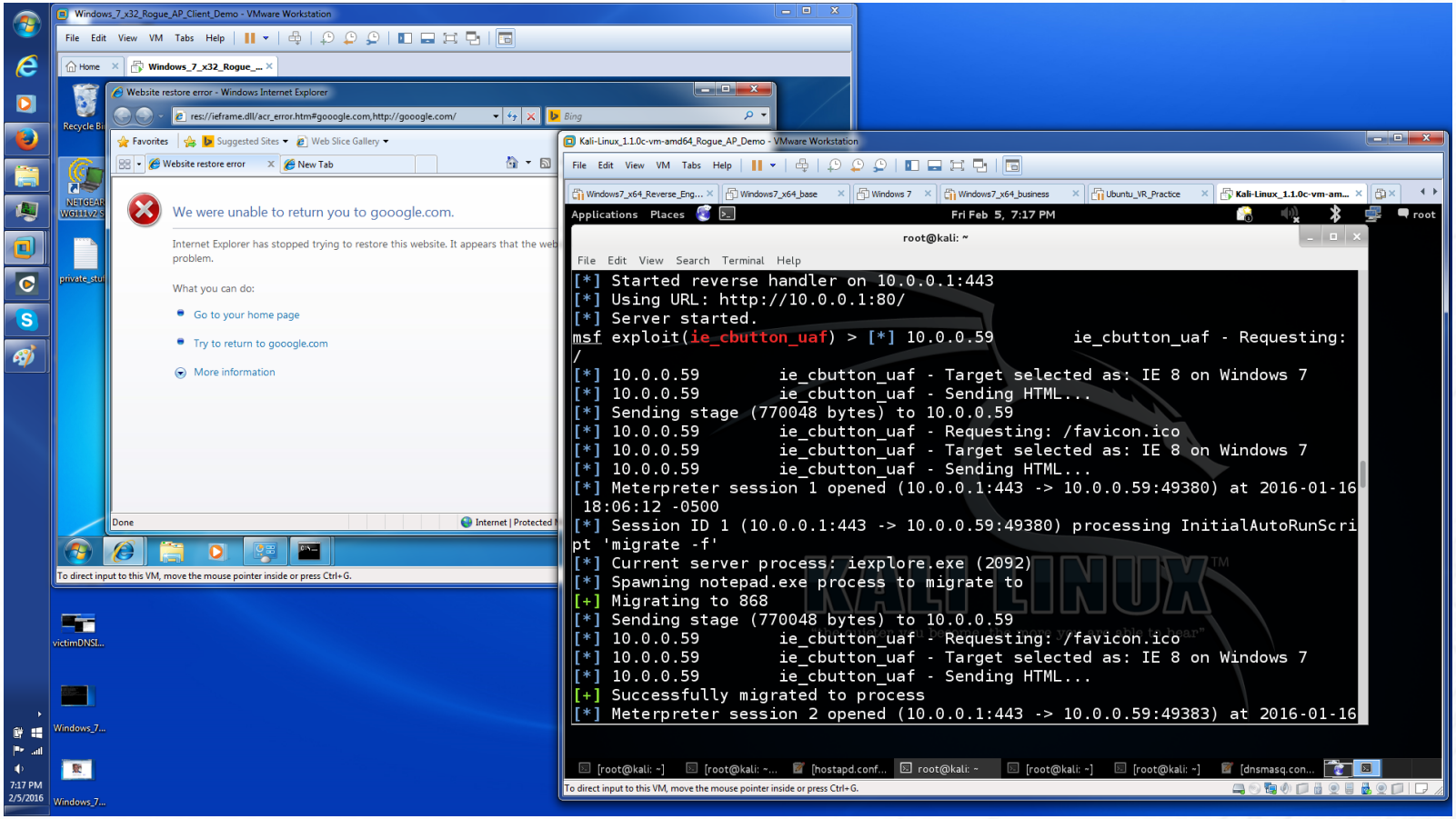
Inject Exploit Code

- 4d : Start the exploit server



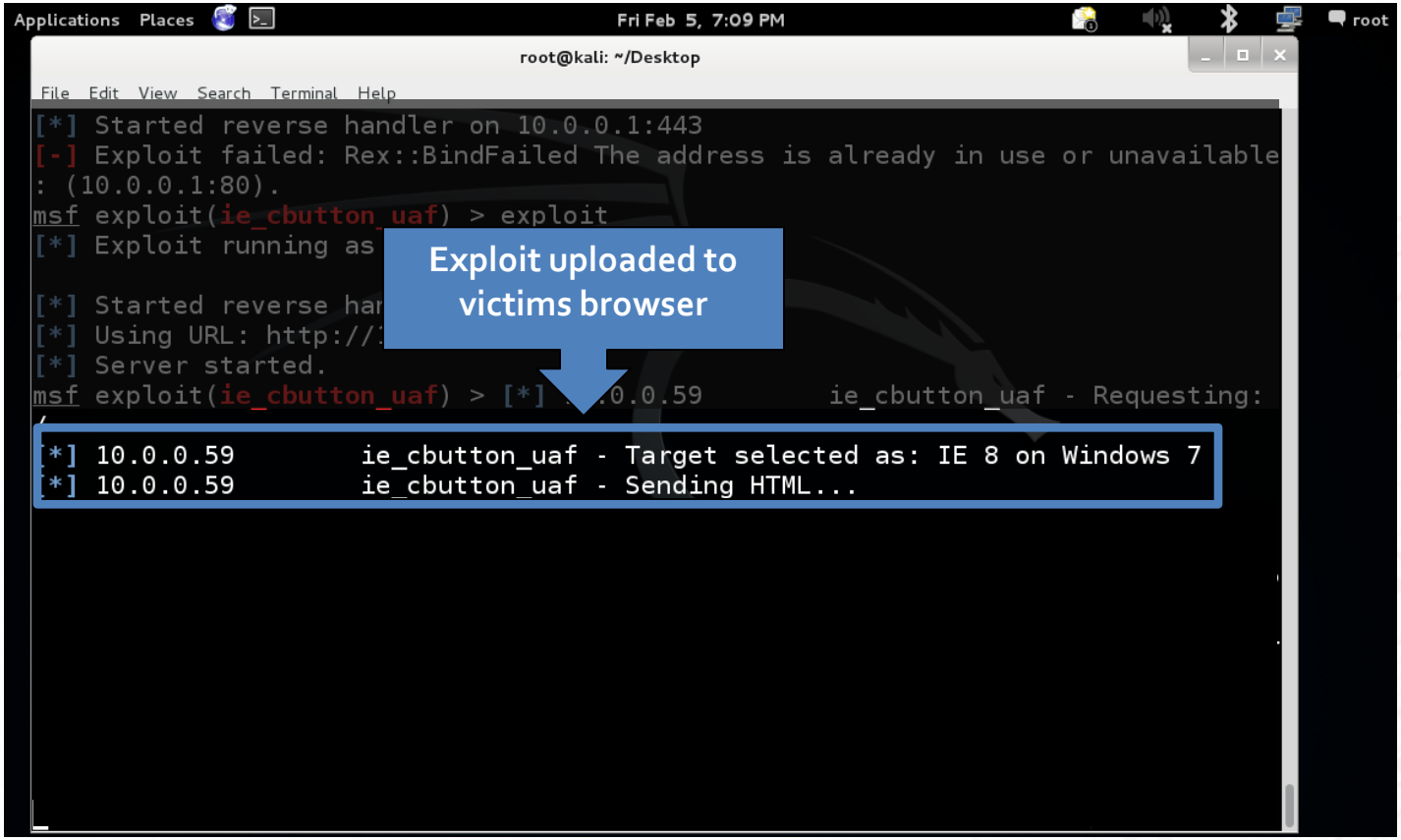
Inject Exploit Code

- 4e : Victim visits gooogle.com



Inject Exploit Code

- 4f : Metasploit server injects exploit code

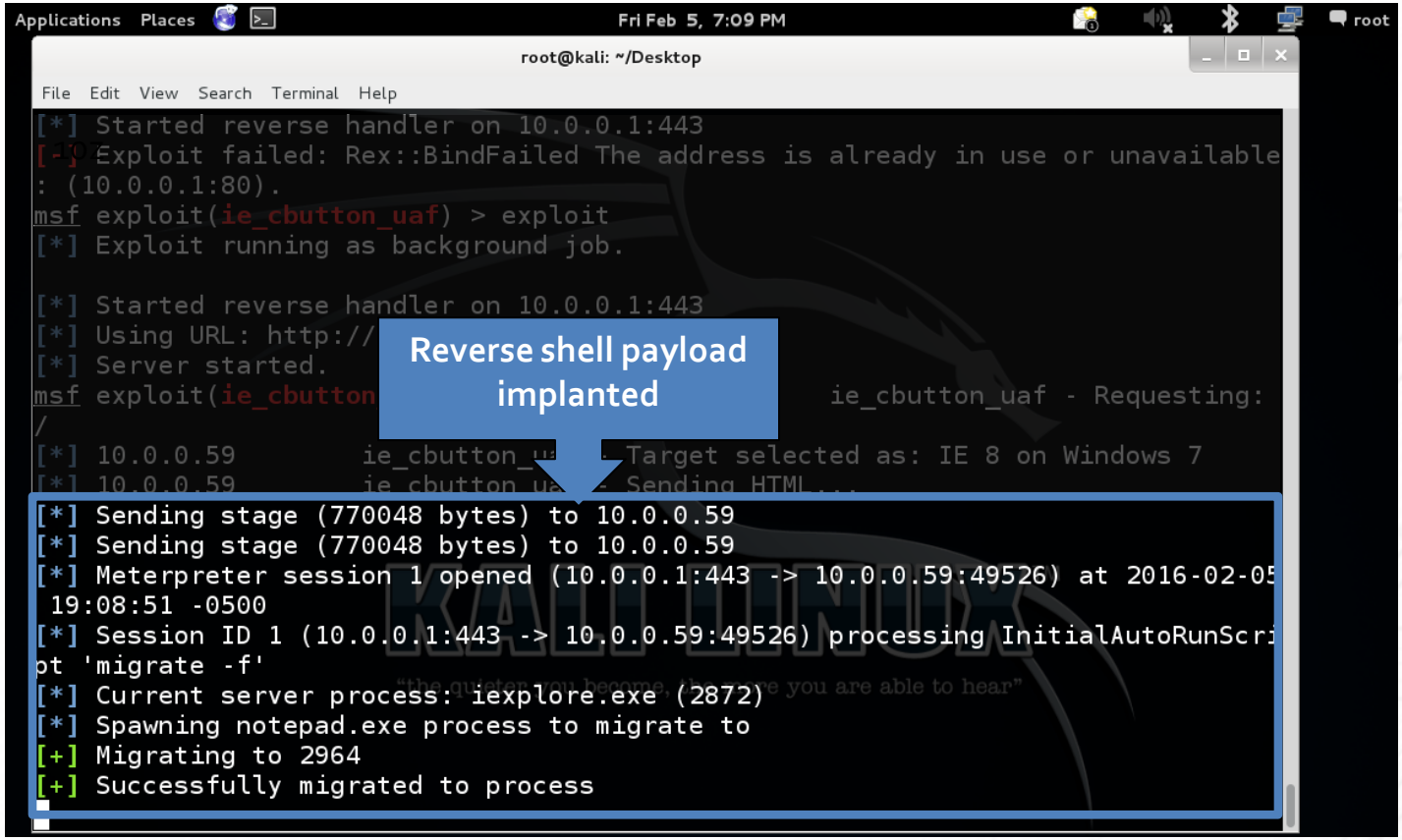


Weaponizing and Deploying Exploits(The Steps)

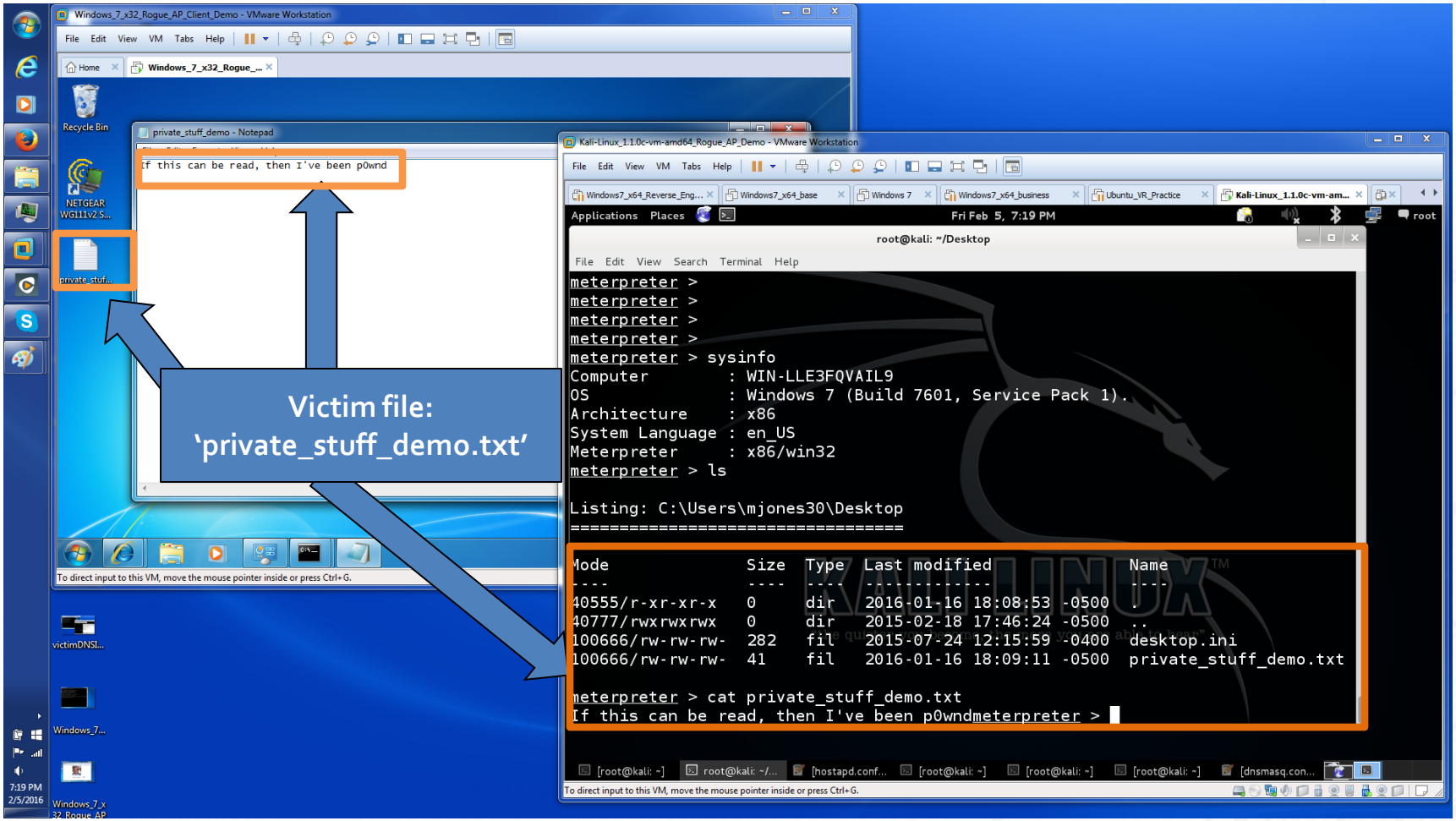
5) Implant malware (e.g. key logger, screenscraper, etc..)

Implant Malware

- 5a : Reverse shell malware implanted on victim

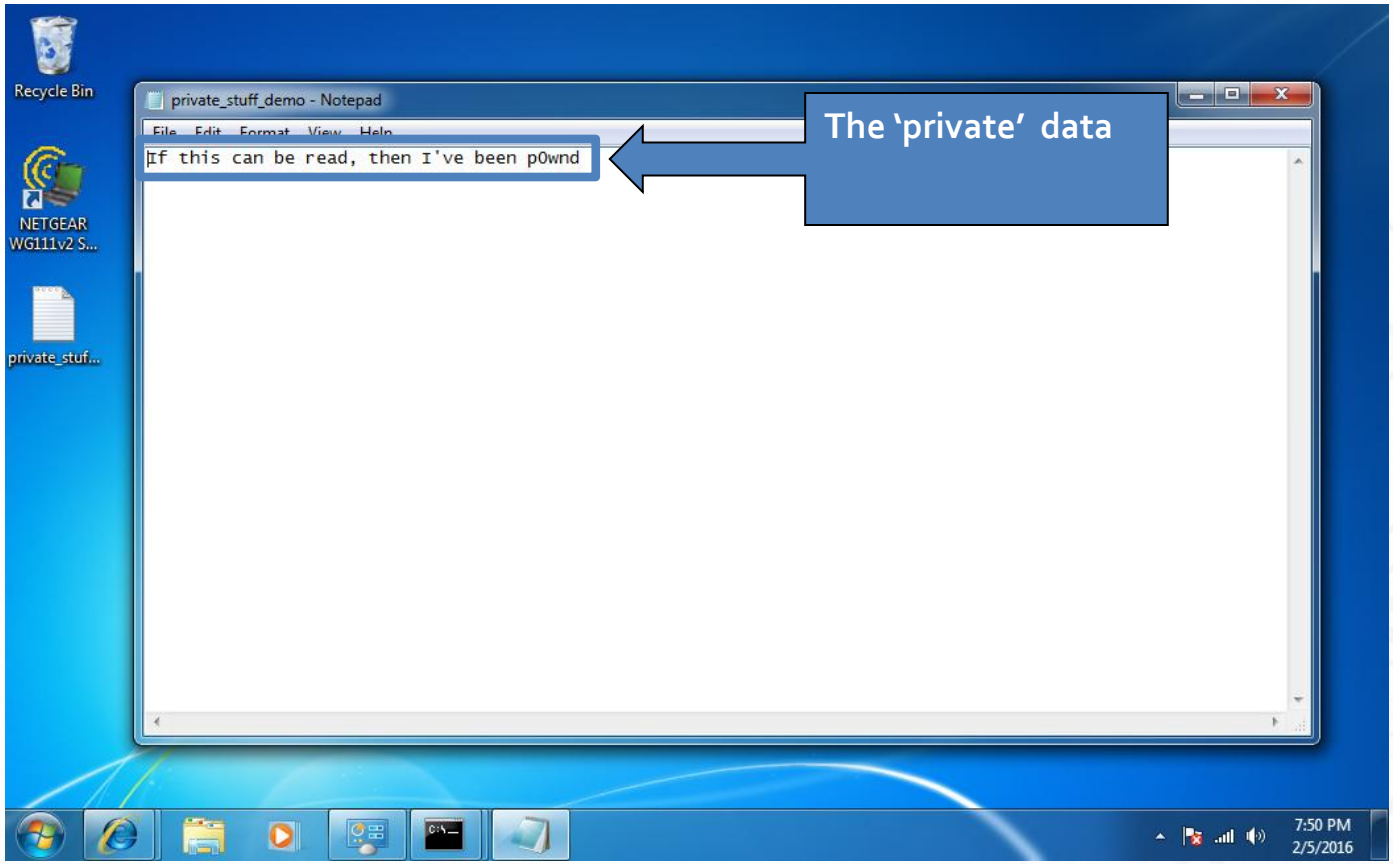


Implant Malware



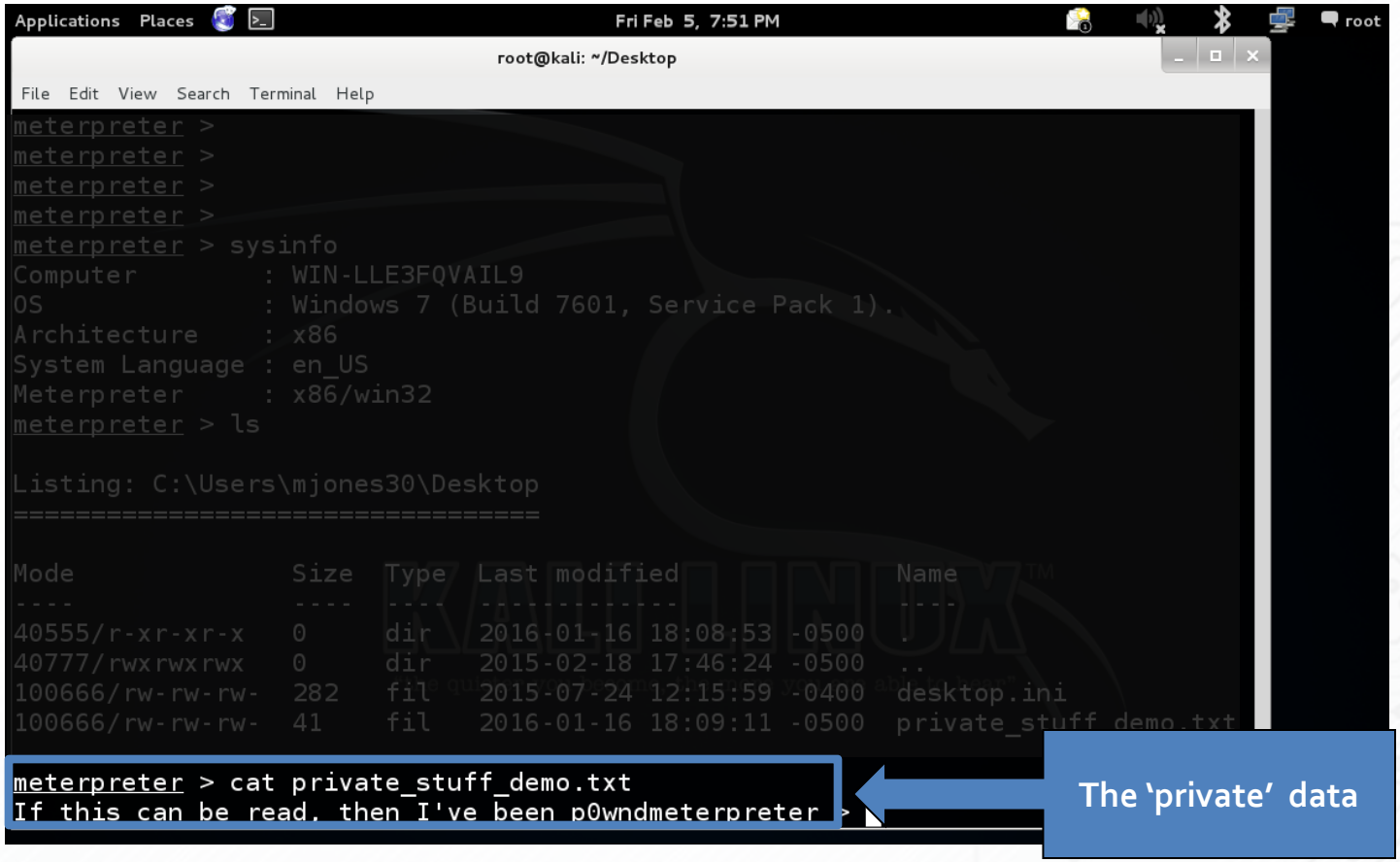
Implant Malware

- 5b :Use shell to access victim data



Implant Malware

- 5b :Use shell to access victim data



Demo: Weaponizing and Deploying

References

1. Joshua J. Drake, et al. (2014). *Android Hacker's Handbook*. Wiley Publishing.
2. Allen Harper, Shon Harris, Jonathan Ness, Chris Eagle, Gideon Lenkey, and Terron Williams. (2015). *Gray Hat Hacking the Ethical Hackers Handbook* (4th ed.). McGraw-Hill Osborne Media.
3. Robert Seacord. (2013). *Secure Coding in C and C++*. Addison-Wesley Professional.
4. Ralf-Philipp Weinmann. 2012. *Baseband attacks: Remote exploitation of memory corruptions in cellular protocol stacks*. In Proceedings of the 6th USENIX conference on Offensive Technologies (WOOT'12). USENIX Association, Berkeley, CA, USA, 2-2.
5. Kleidermacher, D. & Kleidermacher, M. (2012). *Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development*
6. Gebotys, C.H. (2009). *Security in Embedded Devices.* Springer

Questions?

Contact Info

- **Email**: jones_malachi@bah.com
- **LinkedIn**: https://www.linkedin.com/in/malachijonesphd
- **Internships**: cmd_cyber_intern_recruiting@bah.com