Risky Honeypot & Forceful Firewall Final Report

James Barbour, Ben Brooks, Brandon Davis, and Paul Mendoza

Table of Contents

Website exploits

All of our attempted exploits are detailed below. Everything we tried eventually gave us some kind of information, though most took a great deal of trial and error to produce anything. Each section includes what we struggled with as well as how we approached each problem.

Timeline

- Directory traversal
    - Began making educated guesses about directory traversal
    - One such guess was that the downloads would be handled with ?file= get variable
    - Used to get all php files, passwd, etc.
- SQL Injection
    - After much manual trial and error we moved to SQLMAP which was able to uncover the entire database for us
    - Using this we obtained all the user password hashes in secureweb>users>password
- Acquire our account passwords.
    - Wrote script to abuse reset page.
- Acquire API Tokens
    - Wrote script to
- Gain access to playable client

Acquiring TA/Instructor Website Passwords

- PHP file inclusion/directory traversal can be done with http://comp535-lampvm2.cs.unc.edu/download.php?file=/etc/passwd
    - CAN do this on ALL php files
    - See php/ for all our code
    - **Possible Defenses:**
        - Sanitize all requests using paths
        - Use realPath function
        - Ensure path on server starts with a known base path
- user amw can have the password reset, instructions in this paper for guessing captcha:
    - Link to the paper - This was a dead end because of an easier approach
    - Defenses: see section “Acquire Our Group Website Passwords”
- Ryan: possibly something with Rick Astley - failed to get this one
    - Tried running mnemonics against lyrics of “Never gonna give you up”
    - Tried Youtube Video ID for the famous video

- SQLMAP
    - Running the command -"$ *sqlmap - u http://comp535-lampvm2.cs.unc.edu/projects.php?order=schematicsid#Bourne reborn --dbs*" gives us two databases: information_schema, secureweb
    - Command used to get password hashes: "*sqlmap - u http://comp535-lampvm2.cs.unc.edu/projects.php?order=schematicsid#Bourne reborn -D secureweb -T users -C password --dump"*
    - Actual command used for injection point: *order=(SELECT (CASE WHEN (8933=8933) THEN 8933 ELSE 8933*(SELECT 8933 FROM INFORMATION_SCHEMA.CHARACTER_SETS) END))*
    - See SQLMAP LOGS for further info
    - Hashes gained from users table in passwords
    - Used sql queries "SELECT password FROM users WHERE username='<username>'" to match users with hashes
    - **Possible Defenses:**
        - Use prepared statements and parameterized queries
        - Lots of input validation(check for SQL statements)
        - Keep tight privileges on the database
- Found Passwords:
    - solarski : Ap1swa7w
        - Obtained by using kiddough.py on "A picture is sometimes worth a thousand words" and checking against the hash for the user
        - **F594662E1FFAA3AEF92BF59E4E20CC87161AABD0C3A7D7438167E7DCD5AAE5858468CF90FF7EA16E086F22F3761694DE7A4E8562C485CB2F**
    - drdesai : Westborough2013
        - Obtained using 'compositestring.py' and the keywords from deven's fencing page
        - **260897419F7E7CF2B84DBA77C2FBFD1A1A4D24F4D4BF4A1CC96252BC78E0BD3FD0F4CD4B84FF0B69B830CC0E0C228FC0C28BBAC710FD9E56**
    - fabian : fauxtatoes
        - Obtained by applying pronounceable password generation methodology described in the linked paper to the joke in user's description
        - **21F51C0D638ACBFD015716B5BE3CDFBEEAB1ED4F0FF664CC98C71EC0C0815588585B9FA6B959F0D738AC042E34FB7CD6EDF46A4DEED20C17**
    - **Possible Defenses:**
        - Educate users on how to select effective passwords

Acquire Our Group Website Passwords

- Discovered vulnerability in **forgotpw.php**

- We can do an online attack against password reset tokens because the ID of the captcha is stored on the frontend. So we can just script a POST of the form to the server and always complete the same captcha, regardless of which captcha the server sent us. This looks like a pretty big vulnerability that should be exploited.
    - For example POST to /forgotpw.php?user=RiskyHoneypot
        - captcha_name=947
        - captcha_solution=Sy5
        - token=our_token_guess
- Encoding for reset token is base64_encode(md5(seconds::microseconds::username)
    - Example MD5("1492638993::307::ForcefulFirewall")
- Attempt to guess exactly the right microsecond that the server generated the reset token, recreate it, and post it back. This isn't hard because the request RTT from the lab VM is about 4 microseconds. If you just do the same thing over and over, eventually you'll guess correctly. Usually took 3-5 attempts.
- Ran our code from the classroom VM because it is hardwired to lamp and time synchronization was much easier.
- Full implementation in **/website-submission/exploits/fetchpass.py**
- FOUND Passwords:
    - **RiskyHoneypot : YGQVT5Fe0Fh4uQLWiNfnA3hLCyzX0e**
    - **ForcefulFirewall : VHq4w7LJk1RurFSlOyXSAmji6vCiMa**
    - **amw : Nice try.**
- **PROBLEMS WE ENCOUNTERED:** None - this worked as expected.
- **Possible Defenses:**
    - Use "better" randomness to generate the reset token.
    - Do not allow the client to dictate which client it wants to answer - you must put this data in the server database when the captcha is selected.
    - Just don't even store user passwords on the server. Instead, store hashed passwords and use reset tokens to actually allow password resets.

Cracking APIKEY_TOKEN

- Using the information we learned in **header.php** and **account.php**, we knew that we needed to:
    - Figure out which key the server was using by
        - Guessing ALL PIDs
        - Guessing ALL combinations of UID/GID in /etc/passwd and /etc/groups
        - All other values were known (some determined at runtime)
    - Confirm the key by decrypting the APIKEY_TOKEN cookie
    - Encrypt a NEW string "== SecureToken LoggedIn=True Publisher=True == " with the key we discovered.
    - Replace the APIKEY_TOKEN cookie with the encrypted value we generated in order to access a section of code which would perform the API Key lookup in the database for us.

- These steps were automated in **/website-submission/exploits/get_token.php**
- **Problems we encountered:**
    - The tool we developed to automate this attack would have worked much sooner had I noticed that my local system clock was wrong, causing me to guess wrongly about the $_[‘REQUEST_TIME’] component.
    - It took us a few attempts to begin guessing that the USER and GROUP might be different than their primary pairs in /etc/passwd - then start guessing all combinations of users in /etc/passwd and groups in /etc/group
- **Possible Defenses:**
    - This entire user access control model was nonsense. There was no reason to set a cookie at all. The database should simply have checked if the user was a publisher when account.php loads.
    - One could argue that better randomness was needed for the APIKEY_token, but there was never any need to even pass data over to the user.

Data We Found

- ForcefulFirewall passwords: ,xls +mul
- RiskyHoneypot passwords: zjql {ogl
- Use secondary passwords to get over 25 points
    - Obtained these from opening safe 1.
    - **Possible Defenses:**
        - Don’t use a $4 Master Lock
        - Limit access to safe
- Obtained from user solarski
    - `Objects in the game have a velocity for the X direction and the Y direction. Design of Roxanne required us to hard code both of these values. See common_defs.h`
    - **PWD_STORE_token=**TFAK3OkTljkeOzO3OSv1noC4D4CzuQz87MwdnJJTkGqfY6unJwdaR9VPJMSw8cLlzyQjAqIyWk%2Fk3%2BbSZSBczVSzCFKDiBPwwQQDX5OUTEiP0AVlDORpkd0FOvyqiKq1WIGE1x3W36Y1r66JAC5A%2Fkj5jiyED1t5sIVQLD%2BMvnWBIcXtVN3e6HWY9fbnEtuhW%2BFkuUAjDfArdMKYQ2KGw4xoThxjdvjzZH%2BD9DTb9PwhdX5Bx0r3G5lEuA3c4%2FnIuYzdJQ75eEczMbdts4pk0CIpiLA%3D;
    - **PWD_STORE_token_salt=**a%C8%F6;
      SAFE2_TOKEN=5%2B7OPhh9e4%2BDH5YRH1YY7lLgC3N5bYabxf6rVvi8lXrrx%2BeCGo2WC0sSOcXBZSh9
- Obtained from drdesai
    - `I figured that wall hacks will be easy things to accomplish. I have a surprise - if you try wall-hacking, then bots will come for you with vengence! Client objects are described using the data in this enum: include.`
    - **`Same header file as solarski`**

    - **PWD_STORE_token**=Iv%2BdIIA%2FBFBDnkZqEYRzT4H5VWAro%2BdXaKfXZzxD27e6pCOSMAncVIHDu4E%2Fuf2m6%2BtQifi8p%2BPdsaJrW%2BkvyKhK5BZ848P9E5dU6dGdAgqjzuenR0tHdmM1%2BjD89ldDlr8xWkt5PtPy%2FoAq6M8lCWnHvWAsv5brpXabyi0y3FYW9DxFxPQ5u5OaaJiJIJri8UgYKocw6oEzRgYs9bVdbse3%2B9CVgdHzEm6DiDNjGB0UdLfnaqLyoW5mqNanwYl4mZsZvH2bCxGSDYJNwfQ0UNdYUwE%3D;
    - **PWD_STORE_token_salt**=%F1%A8%2A;
    - **SAFE2_TOKEN**=5%2B7OPhh9e4%2BDH5YRH1YY7ILgC3N5bYabxf6rVvi8lXrrx%2BeCGo2WC0sSOcXBZSh9"
- Obtained from fabian
    - `The authentication scheme needed some reliability improvements. The client sends username, password and a hash of the two. Thanks to that, the server just needs to check the hashes. The hash function source is available here: see hash_func.h`
    - **PWD_STORE_token**=9%2B0VbrYYgPFod16jTzoDFWjgxJghj2RbrNsjURuy8%2FFJEGkPeF9opYpRlwFvNQ2YSP3W9I7cxTaPXquNimNlyzB7yiBpiMGi5xrs4q6hL627U2Wt2RMuYAs6CkaTpEDZl%2FKbiiuf%2Bpz3KVm9dTk3%2Fojco7KMgIw057KcTDeJY%2F4O0tpvPjLYquEYkeTPiHKpp1wnKncA2oscTjYWofIgQWnGh8NtpDyzJ08HDYBpngCsVFSfJqW3r4KMD4D%2F44KB%2Fp5RsLesl98gj7y4jpPysWSHxsU%3D;
    - **PWD_STORE_token_salt**=V%2CR;
    - **SAFE2_TOKEN**=5%2B7OPhh9e4%2BDH5YRH1YY7ILgC3N5bYabxf6rVvi8lXrrx%2BeCGo2WC0sSOcXBZSh9
- Obtained from ForcefulFirewall
    - **PWD_STORE_token**=4Qdx1%2BOhiKMJRV9UEAX7nSips%2Bg5JgY0zXKfSQEc9FdZdXQZJ3Qum%2FmQukOHfjVXeOQ9hkGl9Pd2QPbgBedkyUHm8nSg731Z12k4apUskNNjAoy4LM5YFWf7574xuveNOPUSng%3D%3D;
    - **PWD_STORE_token_salt**=%D8%E7%E4;
    - **SAFE2_TOKEN**=5%2B7OPhh9e4%2BDH5YRH1YY7ILgC3N5bYabxf6rVvi8lXrrx%2BeCGo2WC0sSOcXBZSh9
- Obtained from RiskyHoneypot
    - **PWD_STORE_token**=wwZOhzEqJ%2BFtnxCies2zqR%2FPo6pA%2FwdeMJztgIYw4Xeq8YZN2%2Bkn64gXBdnRVNW0%2FTeHLkK3sat%2F5azlgVC91zBMg5S%2Bm0rMyDq3Jqm2rs%2FGpqEtjWxSbTwkZwMELhg106uYuw%3D%3D;
    - **PWD_STORE_token_salt**=%80+9;
      SAFE2_TOKEN=5%2B7OPhh9e4%2BDH5YRH1YY7ILgC3N5bYabxf6rVvi8lXrrx%2BeCGo2WC0sSOcXBZSh9; PHPSESSID=2l8ch53tsd4bh1laqt0rdrjf12
- Shamir secret obtained from gameshop submission:
    - **“DO OR DO NOT THERE IS NO TRY”**

SQLMAP LOGS

Our SQLMAP logs are available in our file dump. **/website-submission/database/**

Reverse engineering CounterStrike

Bypassing Registration

- For the shellcode multiple plans of attack were used to try to bypass the registration key
- First thing that needed to be done was find where the the stack was getting overwritten which was at address **0040CC55**. From here we knew that there was a vulnerability because depending on the length of our file we could overwrite as much of the stack as we wanted.
- Initially it looked like a SEH exploit was needed so a long arduous journey was spent trying to overwrite the last SEH so that we could execute arbitrary code to get pass the registration key.
- After many office hours we took different simpler approach and found that we could just jump directly to the address after the key has been validated: **0040CBDE**
- Things we know:
    - Program asks for file containing 8 characters.
    - Program does not verify there are only 8 character.
    - Bytes 0-7 are read in as the key attempt
    - Bytes 8-12 ??
    - Bytes 13-16 overwrites the address returned to in the program
    - Bytes 64-72 are the next SEH ptr and SEH handler
    - Set bytes 13-16 to the location of key acceptance.

Things We Know About CS Client

- Projectiles:
    - 4124DD defines projectile type
        - 0 - normal shot, 7 damage
        - 1 - goes through walls, 11 damage
        - 2 - red projectile, 20 damage
        - 3 - projectile is yellow, 15 damage
        - 4 - @ symbol, 10 damage
        - 5 - > symbol, heal 3
        - 6 - normal shot?, 7 damage
        - 7 - invisible shot, 9 damage
        - >7 - ? symbol, no damage
    - 4124DF defines y velocity
        - Can set mines, change velocity. Not sure if this behaves like last phase (skips enemies at >1 velocity)

    - 4124E7 defines x velocity
    - Unlimited fire rate by replacing 408EC6 with nop
    - 402B4E fire without pressing anything change to JMP
- Movement:
    - Function at 412CE0
        - 412CFD: delta_y
        - 412D05: delta_x
    - 408a2c and next three functions are wasd handlers

How We Approached CS Hacking

- Finding functions:
    - Initial assumption was that the main game logic/keystroke handler would be in a switch statement
    - First switch statement we found seemed to be display or network-related
    - Second switch statement was core game logic. We tested each case of the switch statement using breaks until we found the portions related to movement and firing
    - We then followed each function call until we found the function calls for both
    - We were able to find the actual API calls for each, where we were able to change things like projectile type and velocity
    - We did not end up doing anything with movement

Our Game Hack

- We have several kinds of hacks all relating to the FIRE logic.
    - Lay several mines at once
    - Fire many projectiles at once in 2 directions
    - Unlimited fire rate
- Both hacks relied on gutting error handling code to make some room to add custom code near address 00408F16. Instead of pushing arguments to the FIRE API from memory, I wrapped the entire call setup in a while loop, and used EDI as a counter (since EDI was unused in that part of the code). These exploits are implemented in various forms in **/counterstrike-round2/modified-clients/**
    - FOR EDI in range -4, 5…
        - Push EDI for each of the X and Y direction arguments
        - Call FIRE api
- For the mine laying hack, we pushed 0,0 for X,Y and kept the loop to lay 10 mines at once. This allowed us to beat the bots by baiting them over our 1 hit kill minepots.
- The unlimited fire rate was as simple as replacing a call to the throttling code (discovered by looking at names and seeing QueryPerformanceCount, then following the call tree up) with a NOP.
- **Problems We Encountered**

    - The hardest thing about these hacks was working with immunity. Every time we saved a new version of the client, we lost all our breakpoints.
    - Immunity was a really painful tool to use.
    - One time, we lost progress because windows started a system update in the middle of working on a hack.

Winning Strategy

- One client armed with mines of multiple supershots (insta kill) would go in first so that they could distract the first bot and kill them on impact when they got close
- Then the second client would go in with the horizontal wall shot/mine combo and cover any other bots that came into the game

Possible Defenses

- Using string safe functions like strncmp instead of strcmp
- Making sure things like ASLR are turned on for the program will significantly hinder being able to refer to arbitrary functions in the code
- Be careful of what libraries/modules are being used in code
- Possibly checking to make sure game logic is being maintained on the server side(checking fire rates and damage being done)

Submission Data

- Our team public keys are in **`/public-keys`**
- Game clients are in **`/counterstrike-round2/modified-clients/`**
- Web exploits are in **`/website-submission/exploits/`**
- Information for running any code will be found in readme files in the code parent folders.