

iCTF 2011

Team **We_Own_YOu**

Vienna University of Technology, seclab

“No plan survives contact with the enemy”

-- *Helmuth von Moltke the Elder (disciple of Carl von Clausewitz)*

Introduction

The annual UCSB International Capture The Flag Contest is the best opportunity for our Students to prove themselves. We announce the contest for students of our Course “Advanced Internet Security”. Recently, this course experienced a massive gain in popularity, because it was included in the Masters colloquium for software engineers. As a result, we can select our participants from a broader range of students with widespread abilities.

Another positive aspect is the date of the event. With our courses starting in October, the students can already be judged based on their midterm results when solving our own challenges. They include the most prominent security-related topics like buffer overflows, web exploits or to some degree reverse engineering. Another aspect is the time setting for the event. With the contest being hosted in California, it usually means 5:00 pm to 2:00 am European time. This is, however, not a problem, except from convincing our security personnel that what we do is nothing illegal. Occasionally, we also had the problem of triggering our University’s IDS, since all traffic was routed through a single port. With the right precautions, however, that proved to be a minor issue.

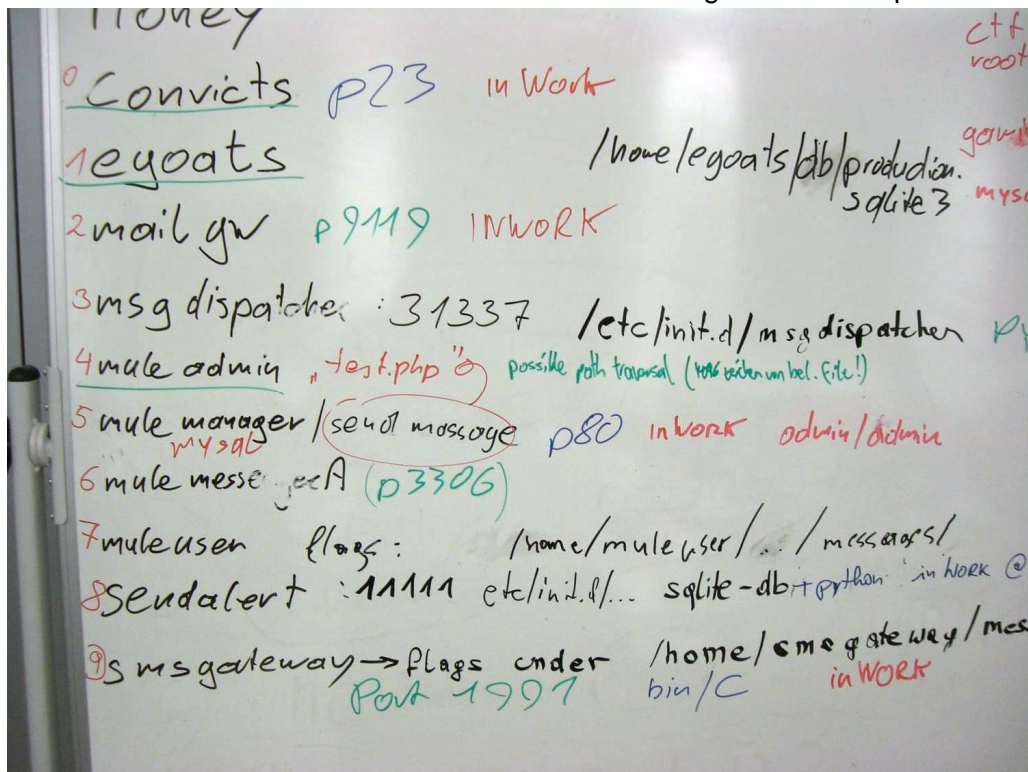
The Team

Our team had the usual structure. It incorporated 20 students from our Advanced Internet Security Course, with the additional requirement that they have our previous course “Internet Security” completed successfully. Furthermore, our lecturers and two masters students completed the team. Most of the organization was done by Adrian Dabrowski and Martin Jauernig, two masters students currently writing their theses at isecLab.

To ensure a wide distribution of skills we asked our students to provide a short summary of their capabilities/interests beforehand. Then, we divided them into different “squads”, according to their skills, with Web, Binary, Network etc. being examples of these mini-teams. During the competition we seated the squad members together to shorten the communication paths.

Previous icfts have shown that it finally boils down to three major problems we had to tackle:

1. Communication: Communication during the contest always proves to be a challenging task. It often happens that students work on the same task without knowing from each other. To soften this impact, we created a samba share where each student was advised to put findings for challenges he is working on /has made progress. For the services, we used a whiteboard, where we put the most relevant information and our current findings for each service. Adrian Dabrowski and Christian Platzer acted as “information hubs”, where the game dynamics were influenced and where high-level decisions were made. In the last two hours for instance, we realized that without solving additional challenges and receiving the money, we had no chance of winning. Therefore, we advised the students to concentrate all their energy on getting us more cash and just maintain the services on their current level without looking for further exploits.



A “screenshot” of our whiteboard after about 3 hours

Communication was still perceived as the major “weak point” in our organisation and we plan to implement some overview tool to display who is working on which problem at what point in the future.

2. Sophistication level: Another lesson learned from previous years was, that there is only so much sophistication one can count on actually implementing. If something is too complicated, only one person can maintain it, and others refrain from looking into it. That is also the reason, why we decided to only set up a very simple MySQL Database

and provide plugins for automatic exploits, the state pusher implementation (once the information was released), the flag submission etc., on an on-demand basis. For a detailed description of our base system see the Infrastructure section. When we talked about the competition afterwards, we came up with dozens of ideas how we could have made our adversaries' life harder (e.g. modifying their flags once we exploited them so the scorebot rates it as down, spending other team's money by poisoning our own flags, etc.). Nevertheless, these ideas need someone to implement it and think of it while actively participating, so most of it was not done during the competition. On the other hand, some of the possibilities strike us as rather radical and we were not sure if it was in line with the fundamental idea behind the competition.

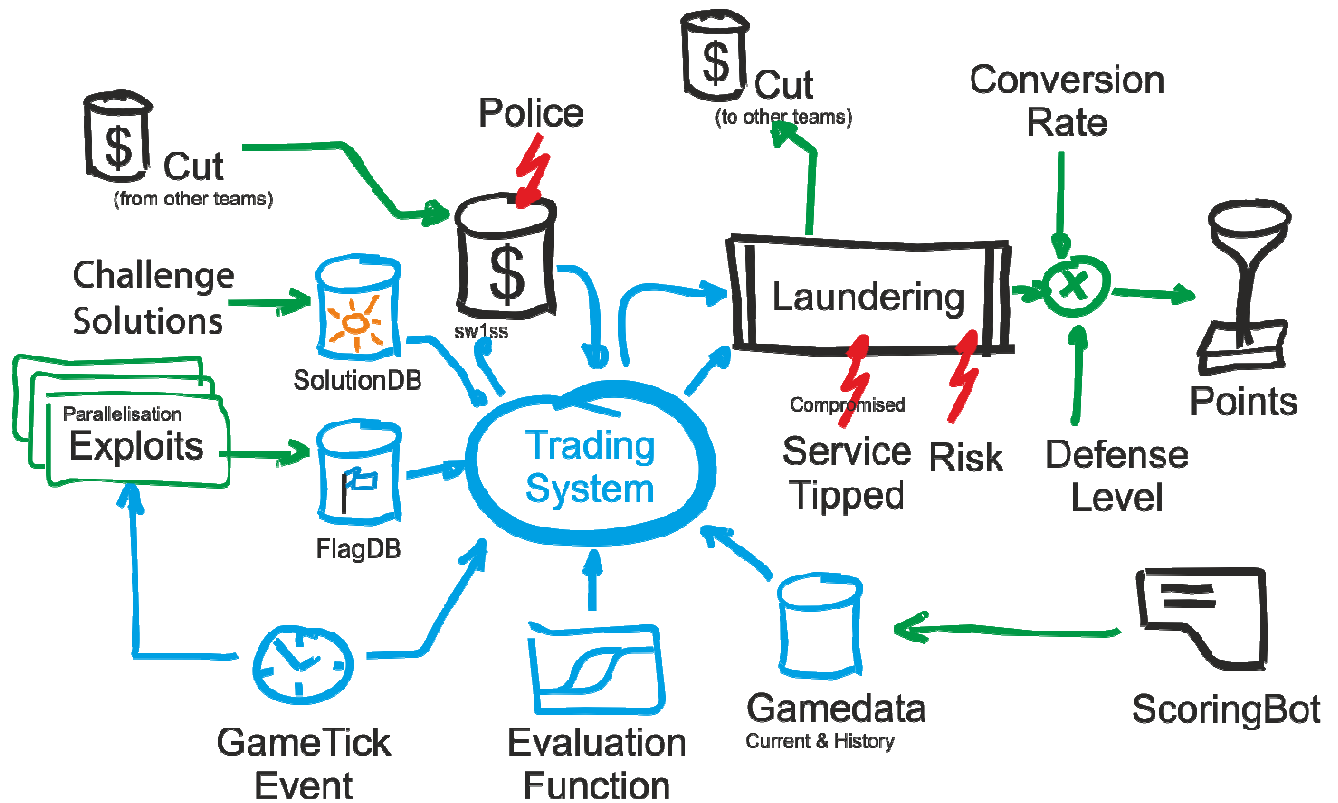
3. Motivation: Finally, motivation is a very essential part of the competition. We instantly tried to convert at least some funds to points to see how flag submissions are handled and what we have to expect later, essentially putting us into first place. That showed our students that we are on the right track and kept us frosty. Additionally, we provided free food, caffeine and sweets to keep every one's brain going. Unfortunately, alcoholic beverages are prohibited to be bought on university budget, hindering us to reach the Ballmer Peak¹. Nevertheless, motivation held throughout the contest, even when we were low on points/money.

Preparation & Infrastructure

Other than in previous years, quite a lot of information was released prior to the contest. As a result, it was suddenly possible to have more concrete topics to discuss than "In general, reversing a binary works like that..." or "If you want to hack some web-app, here are some points to consider...". Instead, we had quite some understanding of the game dynamics and the rather complicated conversion system. On a side-note: The fact that game information was released prior to the competition enhanced the gaming experience tremendously. It was quite a lot of fun to discuss possible strategies, how a framework can be implemented and what the important parts of the environment are.

Three weeks before the ctf, we split the students into four squads: Network & Services, Web, Application and Binary. We determined the shortcomings and held separate preparation sessions with each group. As the description of the game setting were made public, we reviewed it carefully: With game rounds being only two minutes long, we would need a new solution this year. In a "hackathon" two days before the competition we set our game strategy and drafted an automated money-laundering system - that latter proofed essential for our success. Although we were not quite sure, if the whole setup was a big scam, we assumed that it held true during the competition.

¹ <http://xkcd.com/323/>



Initial System design

In the depicted version, we also included a challenge solution submission system. The reason was simply that we thought money at our bank account is bad when the challenge ends. Obviously, this element could be omitted due to the slight game rule changes, which we were quite happy about, by the way. After all, it made the system we had to implement a bit easier.

Our implementation is centered around a simple MySQL Database with several loosely coupled components, like a flag collecting system, a game state source, a laundering evaluation system and an automated flag submission system. This structure allowed us to maintain and test parts of the system without disturbing the rest.

To harvest up to 10 flags from 86 other teams in only two minutes, we built a simple parallel attack framework that automatically submitted the flags to our database. This was the birth of our fifth task force for the game play. Summed up, we had the following elements in place before the competition started:

- The mysql database with some tables to hold all necessary game information (flags, used flags, team info, etc..)
- An internal flag submission system, consisting of a simple php script where flags could be submitted by
`http://gameserver/submit_flag.php?flag=<flag>&team=<teamid>&service=<serviceid>`

- An automated, parallel exploit script that only had to be configured by entering the command line to carry out the exploit and a service number. It queried all available IP addresses from the database and launched 20 parallel processes to carry out the exploit and submit the flag via the aforementioned link. So all that had to be done when an exploit was found was, to enter it into the submission script to fully automate it.
- A game tick synchronization service, that allowed the parallel exploit framework to use the global game time as soon as we were able to derive it from an external source (eg. the game state pusher).
- A firewall setup with a running snort instance, where filter rules (Shorewall) could be implemented on-the-fly, including the required VLAN setup to separate the networks.
- A virtual-box based image host with the snapshot functionality to fall back to running configurations in case we got rooted (which happened once around 10 pm (MET)).

Gamestart

About 45 minutes before the game started, we were allowed to access the rooms designated to host the competition. That proved to be simply not enough. Even though the game pusher's JSON-format was published early this day, we started working in a python-based socket to receive it about 30 minutes before the ctf started.

Getting initial root access to the image proved harder than expected: The default hotkey to break into the bootloader didn't work (we probably didn't try hard enough) so we had to take the long way. The machine consisted of a VirtualBox image with a VMware disk file. VMware tools were unable to loop mount the file system because of the unknown partition entries. After booting with a live CD image into the virtual machine it appeared to be a LVM volume. Luckily, a team member turned out to be firm with that, and we were able to set new passwords. After a reboot we were up and running.

The game-play team also needed some time to adapt to the game state pusher - so we would not have been able to harvest and process flags in this setup period anyway.

In the rush, we connected the image to the wrong VLAN, which voided our carefully crafted firewall rules. That went unnoticed for over an hour. It also meant that our snort installation was dysfunctional.

Services

Students love challenges! It is the most perfidious diversion the iCTF organizers came up with. It is sometimes quite hard to persuade students to take care for a service instead - especially if he or she is the only guy with experience in a particular field (e.g. Ruby on Rails).

The open MySQL port was one of the first exploits we launched. Astonishing enough, we harvested flags from some teams using this vulnerability until the very end of the competition. To exploit this vulnerability, we simply connected to another team's SQL-server using the string `MySQLdb.connect (host = "10.13." + sys.argv[1] + ".3", user = "mulemanager", passwd =`

"grabthiswhileyoucan", db = "mulemanager"). We then selected the row 'address' from the table 'endpoints' and submitted the result to our flag-submission script.

The "Message dispatcher" was quite a simple service, but we were unsuccessful in decompiling and patching the python script. To protect this service we wrote a simple proxy service, that filtered malicious requests on port 31337 and redirected the rest to the original daemon on another secret and firewalled port. We simply changed the listening port number in the .pyc file using a hex editor.

An interesting service we worked on was the 'mulemessageappointment'-service. This turned out to be a compiled python-script which started a web-server on port 8042. The service allowed a user to store and retrieve messages. RSA was used to encrypt these messages. We were able to locate the flags left by the scoring-bot, but we did not manage to completely decompile the pyo-file. We analysed python-assembler code, requests and responses, logs, Subsequently, we were able to identify the private RSA-key as well as the methods used to write and read flags. However, the flags themselves were not only encrypted but also somehow obfuscated, which prevented us from exploiting this service successfully.

Analysis of the 'mule manager'-service quickly uncovered a file called 'test.php'. This script was actually used to steal two flags at once (although we were not sure to which service the other flag belonged, at first). The vulnerable script also allowed us to read arbitrary files. Our attack script was then used to automate an exploit while another member of our team manually started submitting flags, thereby helping us gain an early lead.

A telnet-connection to port 23 on the vulnerable image revealed a service called 'convicts', based on a restricted bash in a telnetd-session. The only binary available for use in that session was a file called 'toilet'. Temporary files created by vim that were found in the same directory allowed us to look at the source-code of that binary. We discovered several vulnerabilities or at least instances of dubious code (a stack overflow with arbitrary payload-size, insecure tmp-files, questionable character filtering and indiscriminate use of `system()`). Additionally, the rbash that served as login-shell proved to be an inadequate security measure. It allowed the user to execute bash-builtin commands (such as echo and read), which enabled us to steal other teams' flags, after the the source-code of 'toilet' revealed their location. Our initial exploit consisted of sending the string `toilet look 'OKTODIGNOW!'; echo excercise_yard/sand/* ; exit;` to extract the names of the files containing the flags followed by: `for i in `grep -a crap_recv1`; do echo "read E < jailcell/toilet/tunnel/`basename $i`; echo \"\$E" >> send2 ; done` to extract the flags.

We were displeased to discover a similar exploit of rbash-functionality in our traffic-dump, which granted root-access to our box. In response to this incident, our first reaction was to block the service. Later, a separate virtual image was set up to isolate this service. We also attempted to build an improved chroot-jail for it. Additionally, we patched some of the more serious problems of 'toilet'. However, the scoring-bot did not honour our efforts by marking the service as 'up', possibly due to latency issues. On the other hand, we noted that this service was 'down' for most teams and still vulnerable where it was operational. This service turned out to be one of the hardest to deal with.

This competition was as much about defense as it was about offense and game strategy. That's why we kept traffic dumps available for all team members. Traffic was dumped on a regular basis and put on our samba share for everyone to access.

We updated a whiteboard (see picture above) with information about all the services, which port, languages, binary paths and locations where they save their flags. This should had helped to recognize opportunities where we can use one service to access flags from another service. Although we had all the information, we missed something that was pointed out to us by an attack against us we captured: Use the Message dispatcher to access the sqlite database of the eGoats service. Exploiting that was quite easy and didn't even require to use the sqlite-shell, as a combination of strings, grep and cut reviled all the flags in this file: `FLAG=`echo "STATUS ../../../../../../home/egoats/db/production.sqlite3" | nc $TARGETHOST 31337 | strings | grep flg | sort | tail -1 | sed -e 's/^.*/flg/flg/' | cut -c 1-39``

The SMS Gateway on port 1991 offered a way to send messages and manage accounts and recipients. One way to get flags was hidden inside the `manage_tcp_client()` function. The name of the sender was used as filename, which content was then returned. If you could get hold of the (random) filename the bot used to store its flag, it was possible to retrieve the flag. Patching this function was quite easy.

Also buffer overflows existed in this service. While researching possibilities for an exploit, we captured an attack of another team, that was using this service for 2-stage attack: They used the overflow to find the file names for the flags (calling `ls messages`), than another method to access their content. We modified the exploit so, that we could read the flags directly in one step. The structure and the sortability of the flags proved to be of great help.

Some exploits that where in development never made it into "production" because we needed to earn more money towards the end.

Challenges

In this section we asked our students to give a short description on how they approached certain challenges. Don't expect it to be complete, it is rather meant as an excerpt of things the participants remembered from those challenges.

Challenge 23: This challenge certainly was inspired by the movie 'Inception' (an archive within an archive/we have to go deeper), or, alternatively, a matryoshka doll. The start: a network dump where parts of a split zip archive are downloaded via http. This archive needs to be unpacked 324 times (with pkzip/gzip/bzip2/7zip/tar, nothing a short perl script can't automate) and yields a '8086 relocatable (Microsoft)' at the end. A simple 'strings' on that file reveals the account number in question.

Challenge 31 and 32: No particular notes here. The reverse-engineering tasks were straight forward and even if it was designed in the numerical order, they were solved backwards (32 before 31), with 31 being not much of a challenge after completing 32.

Challenge 45: This was one of the two challenges solved in the last 10 Minutes (and securing our win). No details were given in the student's description.

Challenge 46: A QR Code was provided to get access a one-time password to access the destination via SSH. It was quite challenging to actually read the password (get the characters right!) and at the same time sync the clock of the android phone which was used to obtain the token.

Challenge 50: Simply google the MD5 sum, every hit on the first result page says it's 'firstblood'. It was solved by most teams during the first few minutes.

Challenge 10: A hidden video frame was displayed near the end of the video clip, showing a lot of beautiful people. Initially we tried to "string" it and search for hidden messages within the stream itself but quickly realized that the message was within the movie itself.

Challenge 15: Why oh Why? A very funny-looking piece of javascript, which displayed a single alert('Why?'). After utilizing a javascript de-obfuscator, the code did not become much clearer, but with better readable variables. After debugging we saw that parts of the code were assembled at runtime, but other parts were not touched at all. We figured that had to be the interesting part, so we put it into the decryption routine which gave us a lot of escaped stuff. Even though we knew it had to be the secret, none of us knew how to un-escape javascript in a fast way, so we just copied parts of it into an alert statement to get to: `obj.secret = "Angelina Jolie's best film in leet speak backwards"`. Nobody believed it to be *Changeling* (which would actually be AJ's best film according to imdb), so our first try, 5r3kc4H, was correct.

Tactics

We had one simple tactic: Never convert money if it's too expensive. As soon as our automatic submission system was in place, we set it to convert money, if the risk was below 20%, the payoff was above 75% and the enemy cut was below 25%. Once we had the complete risk function implemented, we adjusted the *amount* of money to convert just so that we hit the 20% risk level. So instead of calculating risks, we calculated the maximum convertible amount once payoff and cut were within our parameters. Especially when we were low on money (below 400) we raised the conversion bar to 85% payoff and 20% cut.

This is something the Russian team did wrong in our opinion. They got nervous in the last hour, converting their money with bad rates (we roughly calculated it to be around 30-40% in the last peak).

Another important insight was, that the main parts of the risk function are calculated using the amounts laundered so far - not including the currently laundered amount. This meant, that we could convert a large amount every time we got flags from a new service.

Two hours before the challenge ended, we were desperately low on money, even though we had a good number of services exploited and our submission system more or less in place. That's when we decided to concentrate all our efforts on solving challenges to gain money and stopped develop new exploits. Almost too late as we have seen in the last minutes. The final decision fell in the last 10 minutes, when the russian team solved a 150 CHF challenge, and we could score a 300 and one 100 CHF challenge.

Summary

Summed up, we think this year's CTF was one of the best from an organizational point of view. In our opinion, reverting to a peer-to-peer challenge (actually competing with other team) is what adds a lot of value to the competition and was the best decision that could have been made. Initially, we had concerns, that the VPN speed for a 2-minute interval and 87 teams hammering each other's services would inevitably lead to a network-disaster. Fortunately, we were proven wrong.

The host services (flag submission, scoreboard) were functional most of the time, a big plus for motivation and fairness.

A frustrating point was the score bot. Since we did not know if our service was down because the score bot was delayed in querying it or some other team caused mischief, we could not take appropriate countermeasures. We experienced flickering uptimes without noticing any attacks (most of the time affecting 4-6 services at once, which were scored as "up" in the following round). But since every team seems to have experienced this issue, it caused less trouble and evened out throughout the competition. But that's certainly a point to consider in the following icctfs.

The network was amazingly stable. As mentioned above, we did not expect it to work so flawlessly considering that each team could have potentially gathered 870 flags every 2 minutes (Around 75k flags overall?). Our highest regard for the network setup and - maintenance.

And, finally, the balance between the importance of services and challenges was astonishing. Last year, the race was over 4 hours before the competition ended, with almost no way of the second-in-place catching the leader. Now, a team without points but a lot of money was a scary enemy, (as was true for the german team for example). You could never be sure if their tactics was inferior or superior. They could have come up with a 100% conversion rate and take the lead from the last place. Fortunately, they didn't. A very good setting indeed. We look forward to next year.