Security Issue: Memory leak. #2314

Closed **ChinaBluecat** opened this issue on Jul 31, 2020 · 10 comments · Fixed by #2327

ChinaBluecat commented on Jul 31, 2020

Memory leak

This vulnerability happened when the function "SingleDuel::UpdateDeck" and "SingleDuel::PlayerReady" been used.
When the player sends a package with error 'mainc' size and 'sidec' size, the function "SingleDuel::UpdateDeck" haven't check those parameters is legal or not. Then this function will calculate the sum of those parameters.

```
void SingleDuel::UpdateDeck(DuelPlayer* dp, void* pdata, unsigned int len) {
	if(dp->type > 1 || ready[dp->type])
		return;
	char* deckbuf = (char*)pdata;
	int mainc = BufferIO::ReadInt32(deckbuf);               // main_len
	int sidec = BufferIO::ReadInt32(deckbuf);               // side_len
	// verify data
	if((unsigned)mainc + (unsigned)sidec > (len - 8) / 4) {
		STOC_ErrorMsg scem;
		scem.msg = ERRMSG_DECKERROR;
		scem.code = 0;
		NetServer::SendPacketToPlayer(dp, STOC_ERROR_MSG, scem);
		return;
	}
	if(duel_count == 0) {
		deck_error[dp->type] = deckManager.LoadDeck(pdeck[dp->type], (int*)deckbuf, mainc, sidec);
	} else {
		if(deckManager.LoadSide(pdeck[dp->type], (int*)deckbuf, mainc, sidec)) {
			ready[dp->type] = true;
			NetServer::SendPacketToPlayer(dp, STOC_DUEL_START);
			if(ready[0] && ready[1]) {
				NetServer::SendPacketToPlayer(players[tp_player], STOC_SELECT_TP);
				players[1 - tp_player]->state = 0xff;
				players[tp_player]->state = CTOS_TP_RESULT;
				duel_stage = DUEL_STAGE_FIRSTGO;
			}
		} else {
			STOC_ErrorMsg scem;
			scem.msg = ERRMSG_SIDEERROR;
			scem.code = 0;
			NetServer::SendPacketToPlayer(dp, STOC_ERROR_MSG, scem);
		}
	}
}
```

The algorithm thinks 'mainc' and 'sidec' are two unsigned number, so there is an integer overflow, when we set those parameters like (1, -1), their sum will be zero. It is ok though but in the function "deckManager.LoadDeck", it will cause a buffer overread.

```
int DeckManager::LoadDeck(Deck& deck, int* dbuf, int mainc, int sidec) {
	deck.clear();
	int code;
	int errorcode = 0;
	CardData cd;
	for(int i = 0; i < mainc; ++i) {
		code = dbuf[i];
		if(!dataManager.GetData(code, &cd)) {
			errorcode = code;
			continue;
		}
		if(cd.type & TYPE_TOKEN)
			continue;
		else if(cd.type & (TYPE_FUSION | TYPE_SYNCHRO | TYPE_XYZ | TYPE_LINK) && deck.extra.size() < 15) {
			deck.extra.push_back(dataManager.GetCodePointer(code)); //verified by GetData()
		} else if(deck.main.size() < 60) {
			deck.main.push_back(dataManager.GetCodePointer(code));
		}
	}
	for(int i = 0; i < sidec; ++i) {
		code = dbuf[mainc + i];
		if(!dataManager.GetData(code, &cd)) {
			errorcode = code;
			continue;
		}
		if(cd.type & TYPE_TOKEN)
			continue;
		if(deck.side.size() < 15)
			deck.side.push_back(dataManager.GetCodePointer(code));  //verified by GetData()
	}
	return errorcode;
}
```

We can see in this function, the parameters 'mainc' and 'sidec' were treat as two int type numbers. So if we set 'mainc' as '0x7fffffff' and 'sidec' as '0x80000001', in this function it will read 'dbuf' from range 0 to 2147483647.

Function "DeckManager::LoadDeck" also do a important thing: If the memory's data can be found in "dataManager.GetData" function, then it will add it into 'deck'; Else if "dataManager.GetData" function can't find memory's data in database, it will record it in the errorcode and return. Then in the "SingleDuel::UpdateDeck" function, the errorcode will be writen into 'deck_error[dp->type]'.

Now we can see the function "SingleDuel::PlayerReady".

```
void SingleDuel::PlayerReady(DuelPlayer* dp, bool is_ready) {
	if(dp->type > 1)
		return;
	if(ready[dp->type] == is_ready)
		return;
	if(is_ready) {
		unsigned int deckerror = 0;
		if(!host_info.no_check_deck) {
			if(deck_error[dp->type]) {
				deckerror = (DECKERROR_UNKNOWNCARD << 28) + deck_error[dp->type];
			} else {
				bool allow_ocg = host_info.rule == 0 || host_info.rule == 2;
				bool allow_tcg = host_info.rule == 1 || host_info.rule == 2;
				deckerror = deckManager.CheckDeck(pdeck[dp->type], host_info.lflist, allow_ocg, allow_tcg);
			}
		}
		if(deckerror) {
			STOC_HS_PlayerChange scpc;
			scpc.status = (dp->type << 4) | PLAYERCHANGE_NOTREADY;
			NetServer::SendPacketToPlayer(dp, STOC_HS_PLAYER_CHANGE, scpc);
			STOC_ErrorMsg scem;
			scem.msg = ERRMSG_DECKERROR;
			scem.code = deckerror;
			NetServer::SendPacketToPlayer(dp, STOC_ERROR_MSG, scem);
			return;
		}
	}
	ready[dp->type] = is_ready;
	STOC_HS_PlayerChange scpc;
	scpc.status = (dp->type << 4) | (is_ready ? PLAYERCHANGE_READY : PLAYERCHANGE_NOTREADY);
	NetServer::SendPacketToPlayer(players[dp->type], STOC_HS_PLAYER_CHANGE, scpc);
	if(players[1 - dp->type])
		NetServer::SendPacketToPlayer(players[1 - dp->type], STOC_HS_PLAYER_CHANGE, scpc);
	for(auto pit = observers.begin(); pit != observers.end(); ++pit)
		NetServer::SendPacketToPlayer(*pit, STOC_HS_PLAYER_CHANGE, scpc);
#ifdef YGOPRO_SERVER_MODE
	if(cache_recorder)
		NetServer::SendPacketToPlayer(cache_recorder, STOC_HS_PLAYER_CHANGE, scpc);
	if(replay_recorder)
		NetServer::SendPacketToPlayer(replay_recorder, STOC_HS_PLAYER_CHANGE, scpc);
#endif
}
```

This function will check 'deck_error[dp->type]', if it is not zero, it will pack the errcode into 'scem' structure and send it to players. Thanks to it, we can get an easy way to leak the program memory.

And here is my test program, it works well in my local environment.

https://github.com/ChinaBluecat/ygo-fuzz-frame

DyXel commented on Jul 31, 2020

```
==2116==
==2116== More than 10000000 total errors detected.  I'm not reporting any more.
==2116== Final error counts will be inaccurate.  Go fix your program!
==2116== Rerun with --error-limit=no to disable this cutoff.  Note
==2116== that errors may occur in your program without prior warning from
==2116== Valgrind, because errors are no longer being displayed.
==2116==



==2222== HEAP SUMMARY:
==2222==     in use at exit: 70,050,748 bytes in 33,453 blocks
==2222==   total heap usage: 2,230,177 allocs, 2,196,724 frees, 746,782,671 bytes allocated
==2222==
==2222== LEAK SUMMARY:
==2222==    definitely lost: 328,044 bytes in 148 blocks
==2222==    indirectly lost: 473,858 bytes in 1,270 blocks
==2222==      possibly lost: 27,778,656 bytes in 26,019 blocks
==2222==    still reachable: 41,470,190 bytes in 6,016 blocks
==2222==         suppressed: 0 bytes in 0 blocks
==2222== Rerun with --leak-check=full to see details of leaked memory
==2222==
==2222== For counts of detected and suppressed errors, rerun with: -v
==2222== Use --track-origins=yes to see where uninitialised values come from
==2222== ERROR SUMMARY: 95076106 errors from 712 contexts (suppressed: 0 from 0)
```

You are a bit late, this was in 2018, I can only assume the number of leaks have gone up since then.

ChinaBluecat commented on Jul 31, 2020 Author

```
==2116==
==2116== More than 10000000 total errors detected.  I'm not reporting any more.
==2116== Final error counts will be inaccurate.  Go fix your program!
==2116== Rerun with --error-limit=no to disable this cutoff.  Note
==2116== that errors may occur in your program without prior warning from
==2116== Valgrind, because errors are no longer being displayed.
==2116==



==2222== HEAP SUMMARY:
==2222==     in use at exit: 70,050,748 bytes in 33,453 blocks
==2222==   total heap usage: 2,230,177 allocs, 2,196,724 frees, 746,782,671 bytes allocated
==2222==
==2222== LEAK SUMMARY:
==2222==    definitely lost: 328,044 bytes in 148 blocks
==2222==    indirectly lost: 473,858 bytes in 1,270 blocks
==2222==      possibly lost: 27,778,656 bytes in 26,019 blocks
==2222==    still reachable: 41,470,190 bytes in 6,016 blocks
==2222==         suppressed: 0 bytes in 0 blocks
==2222== Rerun with --leak-check=full to see details of leaked memory
==2222==
==2222== For counts of detected and suppressed errors, rerun with: -v
==2222== Use --track-origins=yes to see where uninitialised values come from
==2222== ERROR SUMMARY: 95076106 errors from 712 contexts (suppressed: 0 from 0)
```

> You are a bit late, this was in 2018, I can only assume the number of leaks have gone up since then.

XD, it's so sad to hear that.

DyXel commented on Jul 31, 2020

Yeah, kudos to you for finding this out though. The testing framework also seems like a helpful tool.

DailyShana commented on Aug 2, 2020 Contributor

> This function will check 'deck_error[dp->type]', if it is not zero, it will pack the errcode into 'scem' structure and send it to players. Thanks to it, we can get an easy way to leak the program memory.

I can't understand this. If the `dataManager.GetData` didn't find some data, there would be memory leak?

ChinaBluecat commented on Aug 2, 2020 Author

> > This function will check 'deck_error[dp->type]', if it is not zero, it will pack the errcode into 'scem' structure and send it to players. Thanks to it, we can get an easy way to leak the program memory.
>
> I can't understand this. If the `dataManager.GetData` didn't find some data, there would be memory leak?

"DeckManager::LoadDeck" will check if the code read from dbuf[i] (4 bytes) can be found in the database or not. If it in the database, the function will add it into 'deck'; else will let 'errcode' record it. And 'errcode' will be sent back to players in the "SingleDuel::PlayerReady" function. Also, we can check our deck when we begin a game to know whether there are some dbuf[i] code the same as card ID coincidentally.

DailyShana commented on Aug 2, 2020 Contributor

> "DeckManager::LoadDeck" will check if the code read from dbuf[i] (4 bytes) can be found in the database or not. If it in the database, the function will add it into 'deck'; else will let 'errcode' record it. And 'errcode' will be sent back to players in the "SingleDuel::PlayerReady" function. Also, we can check our deck when we begin a game to know whether there are some dbuf[i] code the same as card ID coincidentally.

So how does it relate to memory leak?

ChinaBluecat commented on Aug 2, 2020 Author

> > "DeckManager::LoadDeck" will check if the code read from dbuf[i] (4 bytes) can be found in the database or not. If it in the database, the function will add it into 'deck'; else will let 'errcode' record it. And 'errcode' will be sent back to players in the "SingleDuel::PlayerReady" function. Also, we can check our deck when we begin a game to know whether there are some dbuf[i] code the same as card ID coincidentally.
>
> So how does it relate to memory leak?

When the player sent a 'CTOS_UPDATE_DECK' package with unsafe 'mainc' and 'sidec', there will be an integer overflow. And that will make function "DeckManager::LoadDeck" read 'dbuf' more then it should be.

```
for(int i = 0; i < mainc; ++i) {
                code = dbuf[i];
                if(!dataManager.GetData(code, &cd)) {
                        errorcode = code;
                        continue;
                }
```

If the parameter 'mainc' is a large number, then 'dbuf[i]' can be far from '&dbuf' location.

DailyShana commented on Aug 2, 2020 Contributor

> "DeckManager::LoadDeck" will check if the code read from dbuf[i] (4 bytes) can be found in the database or not. If it in the database, the function will add it into 'deck'; else will let 'errcode' record it. And 'errcode' will be sent back to players in the "SingleDuel::PlayerReady" function. Also, we can check our deck when we begin a game to know whether there are some dbuf[i] code the same as card ID coincidentally.

I see, it's talking about how to find memory leak

NicoleG25 commented on Dec 1, 2020

Hi,
Is there any plan to address this any time soon?
Please note that CVE-2020-24213 was assigned to this issue.

Thanks in advance !

🚀 1

purerosefallen commented on Dec 1, 2020 Collaborator

I would fix it now. Last time I made a PoC to set `mainc` and `sidec` to `0x7fffffff` and it hacked. Then **@DailyShana** fixed that with a check. However it was not very strict.
How about checking both `mainc` and `sidec` individually again?

purerosefallen mentioned this issue on Dec 1, 2020

fix CVE-2020-24213 #2327

Merged

purerosefallen closed this as completed in #2327 on Mar 16, 2021

jackhong12 mentioned this issue on Jun 7

Broken link ChinaBluecat/ygo-fuzz-framework#1

Closed

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Successfully merging a pull request may close this issue.

fix CVE-2020-24213
Fluorohydride/ygopro

5 participants